A Supervised Learning Approach to Protect Client Authentication on the Web

STEFANO CALZAVARA, Università Ca' Foscari Venezia
GABRIELE TOLOMEI, Università Ca' Foscari Venezia and Yahoo Labs, London
ANDREA CASINI, Università Ca' Foscari Venezia
MICHELE BUGLIESI, Università Ca' Foscari Venezia
SALVATORE ORLANDO, Università Ca' Foscari Venezia

Browser-based defenses have recently been advocated as an effective mechanism to protect potentially insecure web applications against the threats of session hijacking, fixation, and related attacks. In existing approaches, all such defenses ultimately rely on client-side heuristics to automatically detect cookies containing session information, to then protect them against theft or otherwise unintended use. While clearly crucial to the effectiveness of the resulting defense mechanisms, these heuristics have not, as yet, undergone any rigorous assessment of their adequacy. In this paper, we conduct the first such formal assessment, based on a ground truth of 2,464 cookies we collect from 215 popular websites of the Alexa ranking.

To obtain the ground truth, we devise a semi-automatic procedure that draws on the novel notion of *authentication token*, which we introduce to capture multiple web authentication schemes. We test existing browser-based defenses in the literature against our ground truth, unveiling several pitfalls both in the heuristics adopted and in the methods used to assess them. We then propose a new detection method based on *supervised learning*, where our ground truth is used to train a set of binary classifiers, and report on experimental evidence that our method outperforms existing proposals. Interestingly, the resulting classifiers, together with our hands-on experience in the construction of the ground truth, provide new insight on how web authentication is actually implemented in practice.

1. INTRODUCTION

Both HTTP and its secure variant HTTPS, the workhorse protocols of the current World Wide Web, are stateless by design, hence they require web servers to implement their own authentication mechanisms to keep track of state information across different requests. The most widespread solution for tracking state on HTTP(S) relies on *cookies*, i.e., key-value pairs which are chosen by the server so as to identify the user's browser and sent to it. The browser will automatically attach the cookies to any subsequent request to the server: upon receiving back the cookies, the server may use them to pinpoint the requesting client across multiple requests, thus effectively implementing a stateful communication over a stateless protocol.

Modern web applications are complex, and interacting with them involves highly structured communications (i.e., *sessions*), often requiring users to present their cre-

This work was partially supported by the Italian Ministry of University and Research (MIUR) Projects PON TETRis (no. PON01 00451), PRIN CINA (no. 2010FP79LR), PON ADAPT (no. SCN00447), and by the Italian Ministry of Economic Development Project MOTUS (no. MS01_00015 – Industria2015). A preliminary version of this work by a subset of the authors has appeared at WWW'14 [Calzavara et al. 2014].
Author's addresses: S. Calzavara, A. Casini, M. Bugliesi and S. Orlando, Università Ca' Foscari Venezia, Dipartimento di Scienze Ambientali, Informatica e Statistica (DAIS). Gabriele Tolomei is currently affiliated with Yahoo Labs, London, UK.

DOI: http://dx.doi.org/10.1145/0000000.0000000

dentials to log in and authenticate. Correspondingly, the sets of cookies exchanged along such sessions are just as structured and include cookies registered for a variety of purposes: among these, of specific interest for our present concerns are the cookies registered in response to the user presenting her authentication credentials, which we call *authentication cookies*[1].

Authentication cookies are widespread, as they conveniently act as substitutes for the user's credentials during an authenticated session. At the same time (and for the very same reasons), they constitute a primary target of attack, since their inadvertent disclosure may allow an intruder to fully impersonate the user and exploit her privileges in the authenticated session. The complexity of web applications makes room for a large attack surface against authentication cookies, requiring web servers to deploy a variety of counter-measures to achieve a satisfactory degree of protection: for instance, web developers should mark the authentication cookies set by their web applications with the `HTTP-Only` flag, which instructs the browser to prevent any access to them by malicious scripts under the control of the attacker. Unfortunately, as reported in the literature, websites often fail to implement recommended security practices [Zhou and Evans 2010; Nikiforakis et al. 2011; Bugliesi et al. 2014a].

A complementary line of defense, advocated in a series of recent papers, may be built directly within the browser through client-side protection mechanisms [Tang et al. 2011; De Ryck et al. 2012; De Ryck et al. 2011; Bugliesi et al. 2014a; Bugliesi et al. 2014b]. The key idea underlying such mechanisms is to apply the security practices neglected by the server by detecting authentication cookies at the client-side, and enforcing a more conservative browser behavior when accessing them, for instance by applying them the `HTTP-Only` flag when it is not set by the web developers. This process ultimately hinges on an *authentication cookie detector*, i.e., a heuristic which tries to automatically identify the cookies associated with the user credentials among all the cookies stored in the browser, with no support by the user or the remote server. The challenge here is to strike a good balance between *security* and *usability*. On the one hand (*security*) such detector should not miss any authentication cookie, as any miss may leave room for attacks. At the same time (*usability*), they should not over-approximate too much the real set of authentication cookies, since the conservative security policy applied to them may harm the user experience. For instance, if a cookie storing the user's preferences is incorrectly detected as an authentication cookie and marked with the `HTTP-Only` flag, then any legitimate access to it by JavaScript will be forbidden, thus preventing the correct rendering of the website.

Somewhat surprisingly, in spite of their fundamental importance for the effectiveness of the resulting defense mechanisms, none of the heuristics adopted in current systems has, as yet, undergone any rigorous assessment of its adequacy. In fact, existing detectors have so far been evaluated on the basis of intuitive claims assumed as *ground truth* in the evaluation phase (e.g., any cookie whose value is sufficiently long and random is likely used for authentication). Reasonable as they might appear, such claims are obviously biased and, as we show in this paper, hardly adequate for any quality assessment of existing and future client-side defenses.

Contributions. Our first contribution is the design of a (semi-)automatic method to build a ground truth of authentication cookies, i.e., a verified dataset where authentication cookies are isolated and identified correctly. The outcome of this process is a real-world dataset of 2,464 cookies derived from a sample of 215 amongst the today's most popular websites of the Alexa ranking, which we make available for public down-

[1] In Section 3, we define the notion of authentication cookie formally, and make it far more accurate; at this stage, however, the informal characterization given here suffices.

load at `http://bit.ly/1u8Qfiz`. Interestingly, our experience in the construction of the ground truth has unveiled a number of subtleties in the role that different cookies play in web authentication, which appear to have largely been overlooked so far. Based on that experience, we devise a new notion of *authentication token*, which captures multiple web authentication schemes and nicely fits different real-world scenarios. Additionally, as we discuss in the paper, authentication tokens can be used to define a more accurate measure of the protection offered by different security mechanisms built on top of an authentication cookie detector.

Our second contribution is a rigorous evaluation of four existing authentication cookie detectors, based on formal performance measures and conducted against the ground truth we constructed. Our analysis shows a significant degree of misclassification in these detectors, which correspondingly hinders the effectiveness of the client-side defenses built on top of them. Even worse, our data show that the assessment of the existing heuristics is often coarse and ultimately too optimistic, providing a false sense of security. Based on our experiments, we conclude that all the authentication cookie detectors proposed so far are too naïve to be effective in practice.

Our third contribution is, then, the development of a set of *binary classifiers* aimed at automatically and accurately identifying authentication cookies, based on supervised learning techniques. We provide experimental evidence that our proposal outperforms existing solutions, realigning the actual effectiveness of client-side defenses for cookie-based sessions with the optimistic estimations we just mentioned.

Additional Contents. Compared with a previous conference paper [Calzavara et al. 2014], which this work extends, the novel contributions can be summarized as follows:

(1) All the experiments are performed on a much larger ground truth of cookies (2,464 vs. 327) collected from many more websites (215 vs. 70). This allows us to give further and more significant evidence of the problems we identified in our previous work and of the effectiveness of the solutions we proposed therein;
(2) Building a larger ground truth poses new challenges both in terms of computational efficiency and in terms of implementation choices. We discuss a non-trivial optimization of the original algorithm, which runs in linear time (with respect to the number of cookies) for the large majority of the websites and is much faster in practice than the original solution. The new implementation is based on the Selenium[2] Python package rather than on Mechanize[3], since the lack of support for JavaScript in Mechanize prevents the correct functionality of many websites. By using Selenium, we are able to include in the ground truth all the cookies which are set by JavaScript, which we excluded from our previous study;
(3) We experimentally confirm a popular assumption from the literature: the number of authentication cookies which is set by JavaScript is negligible, since we observe that only 4 out of 215 websites (1.9%) use JavaScript to set an authentication cookie;
(4) We formalize a novel measure of protection, which we use to evaluate further the effectiveness of previous heuristics from the literature, as well as our approach;
(5) The larger ground truth we consider in this work is skewed: approximately, only 1 out of 7 cookies is used for authentication purposes. Since the ground truth is skewed, it is more difficult to train an effective classifier. We then revise and make more systematic the machine learning approach to overcome this new challenge and solve the performance problems we observed.

[2]https://pypi.python.org/pypi/selenium
[3]https://pypi.python.org/pypi/mechanize

The rest of the paper is organized as follows. Section 2 provides background about cookie-based session security. Section 3 describes the ground truth construction. Section 4 presents a formal evaluation of existing authentication cookie detectors. Section 5 describes the design of the classifier and contrasts its performance against the state of the art. Section 6 reports on related work, and Section 7 concludes.

2. BACKGROUND: WEB SESSION SECURITY

Cookies are the standard way of implementing a stateful communication paradigm between a client and a server over the HTTP(S) protocol, as showed in Fig. 1 below.

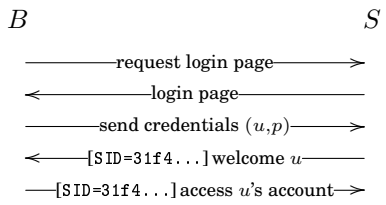

Fig. 1: A typical cookie-based session

The browser B requests a login page to the server S, which requires authentication to access a private area. The user inputs her username u and her password p into the login form, which is then submitted to S. Upon verification of the supplied credentials, the server replies with a welcome page and stores a cookie named `SID` in the user's browser. The value of the cookie is typically a long and random identifier, which uniquely identifies the user's account: since further requests to S will include the cookie, the user will be authenticated with no need to supply her credentials again.

Given that the improper disclosure of the cookie value may allow an attacker to impersonate the user with full privileges, ensuring the confidentiality of authentication cookies like `SID` is a fundamental issue in modern web browsing.

2.1. Cross-site Scripting (XSS)

Web browsers prevent cookies registered by a given domain from being accessed by scripts running on behalf of a different domain, according to the so-called *same-origin policy*. Unfortunately, this simple protection mechanism can be easily circumvented by code injection attacks like XSS, where a script crafted by the attacker runs in the security context of a trusted website [Fogie et al. 2007]. The attack is enabled by a flaw in the input sanitization process by the website, which allows the injection of malicious scripts through the input parameters of its web pages. For instance, assume that the website hosts a search page, which reads a number of keywords from the user and queries a database for results [Nikiforakis et al. 2011]:

```
<?php
  session_start ();
  ...
  $query = $_GET ['q'];
  print "Search results for: <u> $query </u>";
  ...
?>
```

The PHP function `session_start` creates a new authentication cookie, or resumes the current session if a previously generated cookie is attached to the request to the page. The vulnerability arises when the web server generates the result page, since the latter includes the value of the parameter `q` sent by the GET request to the search page. An attacker can steal the authentication cookie just by providing the following link to the victim, e.g., by including it in a malicious web page:

```
http://weak.com/search.php?q=</u><script>
document.write ('<img src ="http://attacker.com/
leak.php?ck =' + document.cookie + '">');
</script>
```

The parameter `q` supplied to the search page is a malicious script, which is injected into the result page; hence, the script runs on behalf of the target website and can access its cookies through the object `document.cookie`, which is sent to the attacker's website as the parameter `ck`.

Since XSS attacks are widespread, web servers can employ the `HTTP-Only` flag to qualify cookies which should not be made available to client-side scripts: `HTTP-Only` cookies will only be accessed by the browser when transmitting HTTP(S) requests to the domain which set them. A few research works suggest to automatically apply the `HTTP-Only` flag to authentication cookies at the browser side when the remote server fails to protect them [Tang et al. 2011; Nikiforakis et al. 2011; Bugliesi et al. 2014a].

2.2. Eavesdropping

Standard web browsers attach all the cookies registered by a given domain to *any* HTTP(S) request transmitted to that domain. Thus, whenever a page loaded over HTTPS retrieves additional contents (e.g., an image) through an HTTP connection to the same domain, the authentication cookies are leaked over HTTP to any attacker who is able to eavesdrop the unencrypted web traffic [Jackson and Barth 2008]. Notice that even websites which are *entirely* deployed over HTTPS may improperly disclose their authentication cookies over HTTP, whenever the attacker is able to inject non-existent HTTP links to these websites in unrelated web pages, since the browser will still try to access these broken links.

The `Secure` flag can be used by web servers to designate cookies that should only be sent over HTTPS connections and never be attached to HTTP requests, thus rectifying the issues above. Similarly to the `HTTP-Only` flag, the `Secure` flag can be selectively applied to authentication cookies at the client-side, thus achieving additional protection against powerful network attackers [Bugliesi et al. 2014a; Bugliesi et al. 2014b]. Another solution against eavesdropping is HSTS [Hodges et al. 2012], a browser security policy which forces any HTTP communication attempt to protected domains to be upgraded to HTTPS. A very recent study recommends the adoption of the `Secure` flag even by websites implementing HSTS, due to a subtle behaviour on sub-domain accesses which can be easily exploited by network attackers to force a leakage of the authentication cookies over HTTP [Kranch and Bonneau 2015].

2.3. Session Fixation

In a session fixation attack, the attacker is able to designate the value of the authentication cookies which will identify the user's session [Johns et al. 2011]. The attacker first gets a set of cookies from the target website and then forces them in the user's browser, for instance by exploiting an XSS vulnerability on the target website. If the website does not refresh the value of its authentication cookies whenever the privilege level of the session changes, i.e., after the user has authenticated by submitting her password, the attacker may be able to impersonate the user just by replaying the orig-

inal set of cookies fixated in the browser. Notice that in this attack the authentication cookies are never leaked to the attacker, but rather he knows them in advance, hence the previous protection mechanisms are clearly bound to fail.

A typical server-side solution against session fixation attacks is to generate a fresh set of authentication cookies upon password checking: since the new cookies will differ from the fixated ones, the attacker will not be able to hijack the session. If the server does not implement this recommended security practice, the attack surface for session fixation can still be significantly reduced at the client-side, by requiring that authentication cookies attached to HTTP(S) requests are only registered through HTTP(S) headers [De Ryck et al. 2012]. The intuition here is that HTTP(S) headers are a much more unlikely attack vector than XSS.

2.4. Preventing Session Hijacking

A recent and interesting web security trend is preventing session hijacking by making useless the knowledge of authentication cookies by the attacker. For instance, one-time cookies [Dacosta et al. 2012] use an HMAC construction to authenticate any request sent by the browser, thus preventing the attacker from replaying a stolen cookie and hijack the session. Similarly, origin bound certificates [Dietz et al. 2012] propose to bind the authentication cookies to the TLS channel used for communication, so that any cookie replay attempt on a different TLS channel will be useless.

Although these solutions are not widely deployed, we do not exclude that existing websites may actually implement some custom technique to mitigate the dangers connected to the disclosure of an authentication cookie by an attacker; however, the real deployment and effectiveness of these countermeasures is very hard to assess, hence properly protecting the authentication cookies at the browser-side is a much safer security practice in the current Web.

3. A GROUND TRUTH OF COOKIES

Building a ground truth of cookies is important to rigorously evaluate the effectiveness of authentication cookie detection techniques and to understand if they actually support a good degree of protection. The construction consists of two steps: (i) collecting sets of cookies from different websites, and (ii) marking each cookie with a binary label to identify the cookie as an authentication cookie or not.

Step (i) has already been recognized as a tedious manual process [Tang et al. 2011]. Indeed, one needs personal accounts on the websites of interest to authenticate to their private areas, which in turn often employ mechanisms like CAPTCHAs to prevent non-human users from registering to them. Step (ii), instead, has been largely overlooked by prior research, while our hands-on experience with web authentication unveils a number of subtle challenges which deserve attention. To tackle these challenges, we introduce the novel concept of *authentication token*, which captures the web authentication schemes adopted by different websites.

3.1. Authentication Tokens

The concept of authentication token is best introduced with an example. As anyone possessing a Facebook[4] account can easily verify, Facebook registers several cookies on the user's browser, for a variety of purposes. From our experience, the only effective way to understand the role of such cookies, and in particular their role in Facebook's authentication mechanism, is to log into the website and observe the effects of deleting the cookies, one by one. Doing that, one notices that deleting either of the cookies `c_user` and `xs` is enough to break the session and get logged out; on the other hand,

[4]http://www.facebook.com

deleting any cookie other than `c_user` and `xs` has no effect on the authenticated session. In other words, both `c_user` and `xs` convey authentication, but neither is enough to authenticate the user. The set $\{\texttt{c\_user}, \texttt{xs}\}$ may thus be identified as the authentication token for Facebook, and `c_user` and `xs` be referred to as authentication cookies.

Generalizing the Facebook case, we define an authentication token as a *minimal* set of cookies which allows the server to authenticate the client, restoring the state of the associated user without asking her to log in again.

Definition 3.1 (*Authentication Token*). Let S be a server and $\mathcal{C}$ the set of cookies it sends to the browser B upon login. We say that $A \subseteq \mathcal{C}$ is an *authentication token* for S if and only if the following conditions hold:

(1) *authentication:* S authenticates B for any request including only the cookies in A;
(2) *minimality:* S does not authenticate B for any request including only cookies in any proper subset of A.

Definition 3.2 (*Authentication Cookie*). A cookie c is an *authentication cookie* iff there exists an authentication token A such that $c \in A$.

Notice that, according to this definition, websites may designate multiple (possibly overlapping) authentication tokens. Indeed, we observed several examples of such authentication scheme in our investigation. For instance, the popular file sharing service Bitshare[5] registers two cookies `PHPSESSID` and `login` on the user's browser, and any of the two is enough to authenticate the user with the website: according to our terminology, the website designates two authentication tokens of size 1.

3.2. Detecting Authentication Cookies

Having introduced a precise notion of authentication cookie, we now devise an algorithm for detecting all the authentication cookies of a given website. The correctness of the algorithm hinges on the observation that web authentication is *monotonic*, i.e., if a set of cookies allows the client to authenticate, then any of its supersets authenticates as well.

3.2.1. Overview and Challenges. Let $\mathcal{C} = \{c_1, \ldots c_n\}$ be the set of all the cookies that server S sends to browser B upon login, we want to identify a *labeling* $\ell : \mathcal{C} \longmapsto \{0, 1\}$ such that $\ell(c_i) = 1$ iff c_i is an authentication cookie. Unfortunately, discovering this labeling boils down to finding out all the cookies included in the authentication tokens registered by S. Since any subset of $\mathcal{C}$ can potentially be an authentication token, and more than one token can occur in $\mathcal{C}$, the search space we have to consider to derive the labeling ℓ is the *powerset* of $\mathcal{C}$, whose cardinality is 2^n, i.e., exponential in the number of cookies. The naïve algorithm for labeling would then be unacceptably slow on many websites; however, we can significantly improve the efficiency of the ground truth construction by designing a smarter exploration of the search space.

First, we observe that, if a set of cookies A is an authentication token, then *all its supersets* authenticate as well, but they can be removed from the search space since none of them can be an authentication token, due to the minimality condition dictated by Definition 3.1. We also notice that, given a set of cookies $\overline{A} \subseteq \mathcal{C}$ such that S does *not* authenticate B, the same happens with *any subset* of $\overline{A}$. This last property resembles the *anti-monotonicity* of the frequent itemsets [Agrawal and Srikant 1994]: if an itemset I is frequent in a transactional database, then any $I' \subset I$ is frequent as well. The *Apriori* algorithm [Agrawal and Srikant 1994] for mining frequent itemsets ex-

[5] http://www.bitshare.com

ploits this property to reduce its exponential search space, by exploring the candidate itemsets from the smallest to the largest.

Similarly to Apriori, a possible solution to our problem could be then to iteratively generate and check subsets of cookies of size k, for increasing values of $k \in [1, n]$: if we enumerate all the subsets of cookies from the smallest to the largest, as soon as we find a subset of cookies that authenticates, we know that it surely is an authentication token, due to the minimality property of the tokens, and we can remove all its supersets from the search space. This algorithm is correct and, despite an exponential worst-case complexity, it is reasonably efficient in practice [Calzavara et al. 2014]. Still, when constructing a ground truth of significant size, like the one we target in this work, it is useful to further refine this idea and come up with a much faster solution.

To improve performances, we exploit a simple observation: even though a general definition of authentication token is needed for formal reasoning and to deal with the complexities of the Web, a large majority of the websites still has only one authentication token (as we experimentally confirm in Section 3.4). For these websites, we can detect the authentication token in *linear* time: intuitively, it is enough to check for each cookie $c_i \in \mathcal{C}$ if the website still authenticates the client when only $\mathcal{C} \setminus \{c_i\}$ is sent to the server; the set of cookies whose removal from $\mathcal{C}$ breaks authentication amounts to the only authentication token. Unfortunately, we do not know in advance if a website has only one authentication token and, if this is not the case, this simple linear algorithm does not work: for instance, if a website authenticates either with $\{c_1\}$ or with $\{c_2\}$, like in the case of Bitshare, deleting cookies one by one from the full set of cookies will never break the session.

3.2.2. The Algorithm. The key to the development of a correct and fast algorithm for ground truth construction is the following lemma.

LEMMA 3.3 (INTERSECTION LEMMA). *Let $\mathcal{C} = \{c_1, \dots c_n\}$ be a set of cookies and let $\mathcal{A} = \{A_1, \dots, A_m\}$ be the (non-empty) set of authentication tokens included therein. Let:*

$$I = \{c_i \in \mathcal{C} \mid \mathcal{C} \setminus \{c_i\} \textit{ does not authenticate the client}\},$$

then $I = \bigcap_{i=1}^{m} A_i$.

PROOF. To prove the result, we must show that $I \subseteq \bigcap_{i=1}^{m} A_i$ and $\bigcap_{i=1}^{m} A_i \subseteq I$:

— Suppose that $c \in I$, we show that c belongs to every A_i, hence $c \in \bigcap_{i=1}^{m} A_i$. Assume by contradiction that there exists A_j such that $c \notin A_j$. This implies $A_j \subseteq \mathcal{C} \setminus \{c\}$, hence we know that $\mathcal{C} \setminus \{c\}$ authenticates. But this implies that $c \notin I$, which is contradictory;
— Suppose that $c \in \bigcap_{i=1}^{m} A_i$, then we know that $c \in A_j$ for each j. This implies that $\mathcal{C} \setminus \{c\}$ does not allow the client to authenticate, hence $c \in I$.

□

Clearly, the set I in the lemma above can be constructed in linear time (w.r.t. the size of $\mathcal{C}$). To understand why the lemma is useful, assume that I allows the client to authenticate: it turns out that I is the only authentication token for the website. Indeed, assume by contradiction that there exist two distinct authentication tokens A_1 and A_2: since $I \subset A_1$ and/or $I \subset \mathcal{A}_2$, but I authenticates, we get a contradiction by the minimality condition of the authentication tokens. Interestingly, the lemma is helpful also whenever I does *not* authenticate the client, since we know that all the authentication tokens must still contain I and we can thus reduce the search space.

Based on this insight, we are finally ready to present the algorithm for finding all the authentication tokens in a given website (Algorithm 1).

Algorithm 1: Detecting Authentication Tokens

Input : A set of cookies $\mathcal{C} = \{c_1, \ldots, c_n\}$ sent by a server S to the browser B upon login
Output: A set of authentication tokens $\mathcal{A}$ for S

```
1  begin
2      I ⟵ buildIntersection(B, S, C);
3      if isAuthenticated(B, S, I) then
4          return {I};
5      else
6          A ⟵ ∅;
7          k ⟵ |I| + 1;
8          cand^(k) ⟵ {I ∪ {c_i} | c_i ∈ C \ I};
9          while cand^(k) ≠ ∅ do
10             foreach C^(k) ∈ cand^(k) do
11                 if isAuthenticated(B, S, C^(k)) then
12                     A ⟵ A ∪ {C^(k)};
13                 end
14             end
15             k ⟵ k + 1;
16             cand^(k) ⟵ Gen&Prune(k, C, A, I);
17         end
18         return A;
19     end
20 end
```

Let B be a browser, S be a server, and $\mathcal{C} = \{c_1, \ldots, c_n\}$ be the full set of cookies sent to B by S upon login. We presuppose the existence of a function $\texttt{isAuthenticated}(B, S, C)$, to check whether the set of cookies C allows B to authenticates at S (a realistic implementation of this function is described in the next section). Using this function, it is straightforward to define a new function $\texttt{buildIntersection}(B, S, \mathcal{C})$, which returns the set I defined in Lemma 3.3 in linear time (w.r.t. the size of $\mathcal{C}$). The main algorithm starts by constructing the set I at line 2. If I allows B to authenticate, the set of authentication tokens includes only I and the algorithm stops, otherwise it generates a set of candidate authentication tokens of size $|I| + 1$ by extending I with one of the cookies which are not already included into it. The algorithm then proceeds by testing each candidate for authentication: if a candidate allows B to authenticate at S, then it surely is an authentication token by the minimality condition and it can be added to the set $\mathcal{A}$ of authentication tokens (line 12). At each further step, the set of the new candidates $cand^{(k)}$ of size k is built by a function `Gen&Prune` (line 16), defined as follows:

$$\texttt{Gen\&Prune}(k, \mathcal{C}, \mathcal{A}, I) = \{C^{(k)} \subseteq \mathcal{C} \mid \nexists A \in \mathcal{A} : A \subset C^{(k)} \wedge I \subseteq C^{(k)}\},$$

where $|C^{(k)}| = k$ and $\mathcal{A}$ is the set of the authentication tokens found so far (of size up to $k - 1$). Indeed, we know that any reasonable candidate must contain I by the intersection lemma, but it must not contain any other authentication token by the minimality condition, hence we can effectively prune the search space. The algorithm stops when no candidate of size k is available and it runs in linear time w.r.t. the number of cookies for any website which has only one authentication token, corresponding to the most

common scenario in the Web. In the worst case, the set I is empty and the algorithm has an exponential complexity w.r.t. the number of cookies [Calzavara et al. 2014].

The output of Algorithm 1 is the set of authentication tokens $\mathcal{A} = \{A_1, \ldots, A_m\}$ for the web server S. To build the ground truth of cookies for S, we define the labeling $\ell : \mathcal{C} \longmapsto \{0, 1\}$ by having $\ell(c) = 1$ iff there exists $A_j \in \mathcal{A}$ such that $c \in A_j$. We repeat the process for each web server S of a given collection of web servers $\mathcal{S}$ to produce the complete ground truth $\mathcal{G}$:

$$\mathcal{G} = \bigcup_{S \in \mathcal{S}} \bigcup_{c \in \mathcal{C}_S} (c, \ell(c)),$$

where $\mathcal{C}_S$ denotes the set of all the cookies that a server $S \in \mathcal{S}$ sends to the browser.

3.2.3. Example. Consider a website registering the set of cookies $\mathcal{C} = \{c_1, c_2, c_3, c_4, c_5\}$. If the website has a single authentication token $A = \{c_1, c_2\}$, the algorithm will start by eliminating the different c_i's one by one, to construct the set $I = \{c_1, c_2\}$ including the cookies whose deletion prevents client authentication. Since the set I authenticates, the only authentication token A coincides with I itself.

Assume now that a second authentication token $A' = \{c_1, c_3\}$ is set by the website. The only cookie whose deletion breaks authentication is c_1: indeed, if c_2 is deleted, then the presence of A' authenticates; if c_3 is deleted, then the presence of A authenticates. Let then $I' = \{c_1\}$, since c_1 alone does not authenticate, the algorithm looks for the authentication tokens starting from I' thanks to Lemma 3.3: A is found at the first iteration, while A' is found at the second one; the other iterations must test $\{c_1, c_4\}$, $\{c_1, c_5\}$, which do not authenticate. Then, the algorithm only needs to test $\{c_1, c_4, c_5\}$, which does not authenticate; all the other subsets must not be tested, since they include at least one authentication token. Hence, the algorithm only explores 10 subsets of $\mathcal{C}$ rather than 32.

3.3. Implementation and Assessment

We implemented the ground truth construction algorithm in Python, using the Selenium package to program a web browser (Firefox) and automate the login process on a list of websites. The script takes as an input a list of triples (w, u, p), where w is the URL of a website homepage, u is a username registered on that website, and p is the corresponding password. For any triple (w, u, p), the script navigates to w, identifies the login form on the page, and submits the credentials (u, p). If the login operation succeeds, the script is given access to a set of cookies $\mathcal{C}$ registered by w and applies Algorithm 1 to detect the authentication cookies in $\mathcal{C}$.

The check performed by `isAuthenticated` (line 8) is implemented as follows. Given a candidate authentication token $C^{(k)}$, we send an HTTP(S) request to w including only the cookies in $C^{(k)}$, and return a positive answer if the response from w satisfies any of the following conditions:

— it does not contain a login form;
— it contains the username u;
— it contains a logout button.

The rationale of this choice is based on common practices implemented by existing websites, which typically display a login form when the user logs out, include in their pages the username associated with the ongoing session and present a logout button to terminate the session. The detection of the login forms and the logout buttons is based on simple heuristics: for instance, we detect a form as a login form whenever it contains a text/email field and a password field. To check the correctness of our implementation and validate the ground truth, we manually accessed all the considered websites from

Firefox and we assessed the output of the script. Specifically, given a set of cookies $\mathcal{C}$ stored in the browser and an authentication token $A \subseteq \mathcal{C}$ identified by our script, we verified that deleting all the cookies in $\mathcal{C} \setminus A$ did not break the session, while removing any cookie in A was enough to get logged out from the website. We noticed that the script was very effective in practice and we incrementally refined the `isAuthenticated` function to make it more precise.

3.4. The Ground Truth: Construction and Analysis

We used our script to crawl 215 popular websites from the Alexa ranking[6], collecting a dataset of 2,464 cookies, including 332 authentication cookies (13.5%). The use of Selenium has proved invaluably helpful to automate the collection process. Indeed, though useful for creating a small ground truth of cookies [Calzavara et al. 2014], a previous version of the script based on the Mechanize library for programmatic web browsing was not robust enough to scale to a significantly larger construction. Specifically, we faced two main problems with Mechanize: (i) many websites were able to detect and prevent a programmatic access by the script, and more importantly (ii) the lack of JavaScript support in Mechanize prevented many websites from working correctly, thus breaking the authentication process. Both these issues are solved by using Selenium, since it allows us to program a standard web browser like Firefox.

Having built a ground truth of cookies, we can finally understand better how web authentication is really deployed in practice and how helpful is our notion of authentication token. We observe that 175 websites (81.4%) only use one authentication token, while the remaining 40 (18.6%) register two tokens; no website sets more than two authentication tokens (see Table I).

Table I: Number of authentication tokens per website

Number of tokens	Number of websites
1	175
2	40

It is hard to understand exactly why a website should set more than one authentication token: an inspection of our data set suggests that some websites are developed using a combination of different frameworks, e.g., PHP and ASP, and apparently more than one mechanism for session management is implemented for generic reasons (e.g., the presence of legacy code). Besides these cases, we think that the only reasonable motivation for deploying more than one authentication token is to decouple authentication from authorization: for instance, mixed content websites which are only partially deployed over HTTPS may use two different cookies to authenticate the user; one cookie is sent over both HTTP and HTTPS and used to keep the authenticated session alive, while the other one is `Secure` and only sent over HTTPS to authorize security-sensitive operations.

Overall, we identified 255 authentication tokens distributed across the 215 websites: 191 tokens (74.9%) contain just one cookie, while 64 tokens (25.1%) are larger. Among these, 53 tokens are composed of two cookies: a manual investigation reveals that a fairly common practice for web authentication is to store the username in one cookie and some random session information in the other cookie. For the remaining cases, it is harder to provide a definite explanation: a reasonable conjecture is that some websites use cookies to store specific session information at the client side rather than

[6] http://www.alexa.com

at the server side; this information is split among different cookies to simplify its retrieval, but it must be entirely available to ensure the functionality of the session. We summarize these numbers in Fig. 2 below.

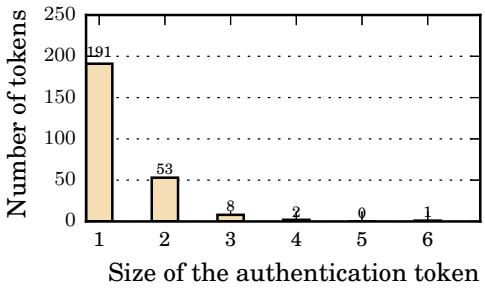

Fig. 2: Size of the authentication tokens

Based on these findings, we argue that, as expected, the most common case for real websites is to just use *the* authentication cookie informally considered in the literature, but the percentage of websites whose analysis benefits from the more general notion of authentication token is not negligible.

Moreover, since the adoption of Selenium allows us to include in the ground truth a number of cookies set by JavaScript, we can concretely verify the popular assumption that "authentication cookies are very rarely set via JavaScript" [Calzavara et al. 2014; De Ryck et al. 2012; Tang et al. 2011]. Experimentally, we detect the cookies which are *not* set by JavaScript by inspecting the headers of the incoming HTTP(S) responses: all the cookies which do not occur therein must be set programmatically. It turns out that only 4 out of 332 authentication cookies (1.2%) are set by JavaScript. All these cookies are set by different websites: `chomikuj.pl`, `vevo.com`, `vube.com` and `nike.com`. After further investigation, we discovered that the last three websites have been developed using AngularJS[7], an increasingly popular framework for developing web applications. We summarize our findings in Table II.

Table II: Authentication cookies set by JavaScript

		Authentication Cookie	
		yes	*no*
JavaScript	*yes*	4	1,026
	no	328	1,106

Finally, we conclude this section with two last remarks on our ground truth. First, even though we automated the construction, clearly one still must possess a personal account on the considered websites to use our script: since the initial registration process is inherently manual, extending the ground truth still requires some human effort. Second, we excluded from the ground truth a number of known cookies set by JavaScript for advertisement purposes (e.g., some cookies set by Google Analytics). All these cookies are uninteresting for our study, since they are never used for authentication purposes and can be ignored by any authentication cookie detector employed by client-side defenses.

[7]https://angularjs.org/

4. EVALUATING CLIENT-SIDE DEFENSES

In this section, we assess the quality of existing state-of-the-art authentication cookie detectors against our ground truth. Despite their importance for the underlying client-side protection mechanisms, all these solutions are quite simple and have been only informally evaluated so far. Instead, we first introduce rigorous validity measures and then we evaluate existing solutions with respect to them: as it turns out, previous evaluations reveal to be way too optimistic, leaving large room for improvement.

4.1. Validity Measures

A standard approach for the evaluation of detectors (or binary classifiers) is based on the count of the number of *true positives* (tp), *true negatives* (tn), *false positives* (fp), and *false negatives* (fn) produced. We refer to "*positive*" as any example in the ground truth that is labeled as an authentication cookie, while we use the term "*negative*" for all the other examples. Hence, tp and tn represent cookies which the detector labels correctly, while fp and fn correspond to mislabeled cookies. Specifically, fp/fn are cookies that the detector labels as positive/negative, but in fact appear as negative/positive in the ground truth. Based on these numbers, we can compute two standard effectiveness measures:

$$specificity = \frac{tn}{tn + fp} \qquad sensitivity = \frac{tp}{tp + fn}$$

Any authentication cookie detector having low *specificity* typically over-approximates the real set of authentication cookies, that is, it makes several fp errors and may lead to usability problems, for instance by marking as `HTTP-Only` some cookies which should be legitimately accessed by JavaScript. Conversely, any solution providing low *sensitivity* leans towards under-approximating the real set of authentication cookies, i.e., it makes many fn errors and leaves room for attacks. We occasionally consider as an aggregate score for an easier comparison also the *F-measure* of a detector, i.e., the harmonic mean of specificity and sensitivity:

$$F\text{-}measure = 2 \cdot \frac{specificity \cdot sensitivity}{specificity + sensitivity}$$

Clearly, not every fp will lead to a usability problem in practice, and not every fn will correspond to a real security violation: understanding these aspects ultimately depends on the semantics of the specific client-side defense. However, the measures above do provide conservative estimation of the effects of deploying a new protection mechanism built over a given authentication cookie detector: as such, we believe it is very important to focus on them in the design phase of new defensive solutions.

Besides the standard measures above, we also find it insightful to exploit our notion of authentication token to get an additional measure of protection. Intuitively, a client-side defense for cookie-based sessions is effective whenever its underlying authentication cookie detector is able to identify at least one authentication cookie for each authentication token. If this minimal set of authentication cookies is safeguarded, we are guaranteed that the website is protected against session hijacking[8].

Formally, let W be the full set of websites where a detector D is tested. For a website $w \in W$, let $\mathcal{A}_w$ stand for the set of authentication tokens of w and let $tp_D(w)$ be the set of true positives produced by D on w. We define the *protection* granted by D on W,

[8]It may actually be the case that the value of the protected cookies could be easily predicted from the value of the unprotected ones, but this exclusively depends on the specific implementation of the web application and a corresponding protection measure would be pretty hard to define in the general case.

written $\rho_D(W)$, as follows:

$$\rho_D(W) = \frac{|\{w \in W \mid \forall A \in \mathcal{A}_w : tp_D(w) \cap A \neq \emptyset\}|}{|W|}$$

Even though previous works [De Ryck et al. 2012; Tang et al. 2011] already tried to estimate the number of protected websites, their conclusions are not quite accurate, as we elaborate in the following through the study of our ground truth.

4.2. (Re-)Evaluating Existing Solutions

We focus on the authentication cookie detectors proposed by the following four solutions published in the literature:

— SESSIONSHIELD [Nikiforakis et al. 2011] is a proxy between the browser and the network which aims at protecting cookie-based sessions against XSS attacks. The core idea is to automatically detect and store the authentication cookies in a private database, inaccessible to JavaScript, thus emulating the browser behavior implemented for the `HTTP-Only` flag;
— SERENE [De Ryck et al. 2012] is a client-side solution against session fixation attacks. It applies a detection algorithm to spot cookies which are likely used for authentication, but have not been registered through HTTP headers. Since these cookies can be potentially fixated by a malicious script, they are stripped away from HTTP requests and never used for authentication;
— COOKIEXT [Bugliesi et al. 2014a] is an extension for Google Chrome designed to ensure the confidentiality of authentication cookies against both XSS attacks and network eavesdropping. COOKIEXT selectively applies both the `HTTP-Only` and the `Secure` flag to authentication cookies, while forcing a redirection from HTTP to HTTPS for supporting websites;
— ZAN [Tang et al. 2011] is an extension for the OP2 web browser aimed at protecting legacy web applications against different kinds of vulnerabilities. Notably, ZAN automatically applies the `HTTP-Only` flag to the authentication cookies it detects, to prevent their leakage via XSS.

All the authentication cookie detectors adopted by these tools are based on the same empirical observation: authentication cookies are typically longer and "more random" than other cookies, and they often contain authentication-related words in their keys (e.g., "*sess*"). Though the various tools differ in several aspects (for instance, randomness can be estimated in several ways, and different weights and thresholds can be empirically set for the same feature), we abstract from the details here, and we just summarize the aspects that are most relevant to our present needs. For any further information, we refer to the original papers cited above. Similarly, we omit details about some standard measures of randomness: Shannon entropy [Shannon 1948], index of coincidence [Friedman 1922] and password strength [Florêncio and Herley 2007].

Table III provides an intuitive description of the features used by each detector to mark a cookie $c = (k, v)$ as an authentication cookie.

In Table IV, we represent each detector as a boolean formula $\phi(c)$, which holds true if and only if the cookie c is recognized as an authentication cookie.

For instance, COOKIEXT marks $c = (k, v)$ as an authentication cookie whenever k contains some specific authentication-related words, or v has a low index of coincidence and is long enough (the thresholds have been empirically set).

4.2.1. Criticisms to Previous Assessments. The assessment of existing authentication cookie detectors has so far been organized around (i) the collection of a dataset of cook-

Table III: Features used to label the cookie $c = (k, v)$ as an authentication cookie

Check	Description	SHIELD	SERENE	COOKIEXT	ZAN
$auth(k)$	k contains authentication terms (e.g., "*sess*")	✔	✔	✔	✔
$known(k)$	k is a standard name (e.g., `PHPSESSID`)		✔		
$len(v)$	length of v is above a threshold	✔	✔	✔	✔
$H(v)$	Shannon entropy of v is above a threshold				✔
$IC(v)$	index of coincidence of v is below a threshold			✔	
$s(v)$	password strength of v is above a threshold	✔	✔		
$dict(v)$	v matches a dictionary word	✔	✔		

Table IV: Checks to label the cookie $c = (k, v)$ as an authentication cookie

$$\text{SHIELD}(c) \triangleq (auth(k) \wedge len(v)) \vee (s(v) \vee \neg dict(v))$$
$$\text{SERENE}(c) \triangleq known(k) \vee (auth(k) \wedge s(v) \wedge \neg dict(v) \wedge len(v))$$
$$\text{COOKIEXT}(c) \triangleq auth(k) \vee (IC(v) \wedge len(v))$$
$$\text{ZAN}(c) \triangleq (auth(k) \wedge (H(v) \vee len(v))) \vee (\neg auth(k) \wedge H(v) \wedge len(v))$$

ies from existing websites, followed by (ii) a manual investigation aimed at estimating the number of fp and fn produced by the detector.

This approach suffers from two fundamental flaws. First, and most importantly, the manual investigation is not carried out by authenticating to the considered websites, but rather by inspecting the structure of each cookie marked positive (or negative) by the detector: misclassifications are then *estimated* based on the expected syntactic structure of a standard session identifier. For example, any cookie containing a long random value which is marked negative by the detector is considered a fn (the dual reasoning leads to an estimate of the number of fp). Clearly, this approach is convenient to carry out in practice, but provides a very coarse and overly optimistic estimation, which is ultimately biased by the idea of assessing the effectiveness of the detector against the very same observations underlying its design.

The second flaw in existing assessment methods is that, in general, the number of fp and fn is not a statistically significant measure of the performance of a detector, since the distribution of the correctly labeled instances (tp and tn) cannot be ignored [Mitchell 1997]. This is particularly important when the classes of positive and negative examples are highly skewed like in our setting, where authentication cookies are far less than non-authentication ones. Unfortunately, the lack of any precise report about the number of tp and tn produced by previous detectors prevented us from computing specificity and sensitivity for existing evaluations, which is something we would have liked to consider for further comparison.

4.2.2. Results of Our Own Assessment. We evaluate each of the four authentication cookie detectors by constructing a corresponding *confusion matrix*. This is a 2×2 table, where tp and tn are given in the main diagonal, while the antidiagonal contains fn and fp. We start with the detector adopted by SESSIONSHIELD, whose confusion matrix is shown in Table V.

Table V: Confusion matrix for SESSIONSHIELD

		Predicted	
		positive	*negative*
Actual	*positive*	tp: 297	fn: 35
	negative	fp: 1,226	tn: 906

We notice that the sensitivity is impressive (89%), since the algorithm produces only 35 fn. Indeed, our protection measure reveals that the 99.5% of the websites are pro-

tected! On the other hand, the specificity is very low (42%), as 1,226 fp are produced by the detector on our dataset of 2,464 cookies. This strongly conflicts with the preliminary evaluation performed by the authors of SESSIONSHIELD, who estimated only 19 fp in a set of 2,167 collected cookies: the difference between the informal estimation above and the formal evaluation we conduct in this work is of two orders of magnitude. Interestingly, the authors of SERENE initially tried to reuse the authentication cookie detector employed by SESSIONSHIELD, but eventually realized they needed a much more accurate solution to deal with several usability issues, due to a rather high number of fp identified in practice [De Ryck et al. 2012]. We conclude that the detection algorithm of SESSIONSHIELD is too inaccurate to live up to any large-scale usability study of a client-side defense based on it.

Next, we turn to SERENE, whose confusion matrix is given in Table VI below.

Table VI: Confusion matrix for SERENE

		Predicted	
		positive	*negative*
Actual	*positive*	tp: 126	fn: 206
	negative	fp: 336	tn: 1,796

The number of fp significantly decreases (336 vs. 1,226) with respect to the authentication cookie detector of SESSIONSHIELD and the specificity greatly improves (84%). On the other hand, we notice that only 126 out of 332 authentication cookies are successfully recognized, and the sensitivity of the detector is really low (38%). In the original paper on SERENE [De Ryck et al. 2012], the authors empirically estimated that their solution is able to protect the 83.4% of the considered websites. However, by evaluating SERENE against our ground truth, we discover that the tool would be able to protect only the 41.4% of the websites.

To be fair, we also evaluate COOKIEXT, a solution proposed by two of the authors of this paper. Similarly to what happened for the other proposals, we acknowledge that our initial evaluation of COOKIEXT provided some overly optimistic results. We report in Table VII the result of our later assessment against the ground truth.

Table VII: Confusion matrix for COOKIEXT

		Predicted	
		positive	*negative*
Actual	*positive*	tp: 192	fn: 140
	negative	fp: 494	tn: 1,698

The protection offered by COOKIEXT appears significantly stronger than the previous one, since the number of fn produced by the detector is much lower (140 vs. 206) and the sensitivity increases (58%). Still, the degree of protection is lower than our original estimate, and certainly needs improvement: indeed, our informal estimate in the COOKIEXT paper identified only 29 potential fn in a set of 2,291 collected cookies. The difference between the formal evaluation and the informal estimation is then of one order of magnitude and the impact on security is significant. By computing the protection measure, we observe that COOKIEXT is able to protect the 64.2% of the websites, something which we did not consider in the original paper.

We conclude with ZAN, whose confusion matrix is given in Table VIII.

Again, we have a considerable number of fn, since 141 out of 332 authentication cookies are misclassified, and the sensitivity is not very satisfactory (58%). However, the original paper on ZAN did not consider the accuracy of the authentication cookie

Table VIII: Confusion matrix for ZAN

		Predicted	
		positive	*negative*
Actual	*positive*	*tp*: 191	*fn*: 141
	negative	*fp*: 443	*tn*: 1,689

detector itself, rather it provided an evaluation about the gain in protection enabled by the tool through an analysis at the website level. In particular, the authors stated that 103 out of 136 considered websites (75.7%) were successfully protected by ZAN: all the 33 failures were due to the underlying detection algorithm. If we compute the protection of ZAN, however, we discover that only the 65.1% of the websites are protected. The difference is less significant than in the previous cases, but still worth noticing.

To conclude, as we summarize in Fig. 3, we argue that the only authentication cookie detector which presents a satisfactory degree of protection is the one employed by SESSIONSHIELD, which safeguards the 99.5% of the websites. Unfortunately, that detector has very low specificity and has proven impractical when initially implemented in SERENE; based on our measurements, we actually argue that it would not be fit for any practical client-side defense. For all the other detectors, our evaluation highlights an unexpectedly low degree of protection, which significantly limits their effectiveness, since at most the 65% of the websites are protected.

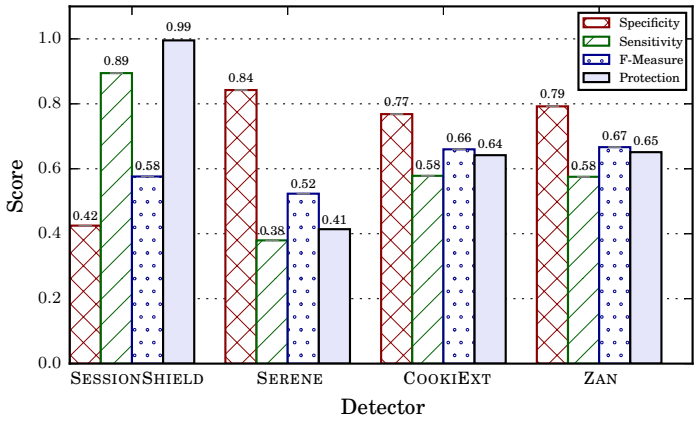

Fig. 3: Evaluation of the different detectors

5. IMPROVING CLIENT-SIDE DEFENSES

Given the inaccuracy of existing solutions, we leverage machine learning techniques to devise a novel authentication cookie detector, which is able to provide a high degree of protection, while being precise enough to be usable in practice. We emphasize that the existence of such a detector is not obvious at all, mainly due to a number of inconsistencies across different websites, which make the authentication cookie detection problem difficult to solve even by manual inspection: this is one of the reasons why previous informal evaluations turned out to be imprecise and ultimately unreliable.

5.1. A Supervised Learning Approach

The problem of automatically discovering authentication cookies can be reduced to an instance of the *binary classification* problem: the goal is to synthesize an algorithm which is able to effectively discriminate cookies that are part of at least one authentication token from those which are not. To uniform with standard terminology, we see authentication and non-authentication cookies as two separate *classes* of interest. We refer to any observed cookie whose class membership is known as a *labeled instance*; the classification problem exploits a sample of labeled instances, called *training set*, in order to *predict* which class a new (i.e., out-of-sample) instance belongs to.

Usually, each instance is represented by a set of individual, measurable properties of the observation called *feature vector*, and the resulting vector space is referred to as *feature space*. More specifically, in our setting we denote by X the cookie feature space, and we represent each cookie c with a d-dimensional feature vector $\mathbf{c} = (x_1, \ldots, x_d)^T$, where each feature x_i describes a particular property of the cookie. We presuppose a *feature extraction* function $\chi : \mathcal{C} \longmapsto X$, which transforms each cookie from the population $\mathcal{C}$ into its corresponding feature space representation.

Given a ground truth $\mathcal{G}$, we can construct a sample training set $\mathcal{D}_{\text{train}}$ by picking some $\mathcal{G}_{\text{train}} \subseteq \mathcal{G}$ and by transforming each cookie occurring therein with its feature space representation. Formally, we let:

$$\mathcal{D}_{\text{train}} = \bigcup_{(c,\ell(c)) \in \mathcal{G}_{\text{train}}} (\chi(c), \ell(c)).$$

A *supervised learning* algorithm infers from $\mathcal{D}_{\text{train}}$ a function $\lambda : X \longmapsto \{0, 1\}$, which maps feature vectors into an expected class label. The corresponding binary classifier is then the function $\lambda \circ \chi : \mathcal{C} \longmapsto \{0, 1\}$, which predicts the class of an arbitrary cookie from its feature space representation. Clearly, the choice of different learning algorithms will lead to different classifiers, and among these we want to choose the most accurate one; it is well-known that different approaches may perform well or poorly, depending on the specific application task [Brodley and Utgoff 1995; Perlich et al. 2003]. We refer the interested reader to Appendix A.1 for a discussion on the main challenges of supervised learning in our setting.

5.2. Choosing the Best Features

An important design choice we make is considering only features which can be computed from the set of cookies registered by a given website through a single HTTP response. This is the key to make the classifier work properly in practice, e.g., when it is included in a browser extension aimed at protecting authentication cookies. We explore not only well-known features that were already proposed in the literature (Section 5.2.1), but also a novel class of *contextual* features (Section 5.2.2). We report on our most interesting findings, by evaluating the capability of these features to properly discriminate between authentication and non-authentication cookies in our ground truth.

5.2.1. Non-contextual Features. We first started by exploring the effectiveness of some features illustrated in Table III, and exploited by authentication cookie detectors presented in the literature. All these features are extracted from a single cookie $c = (k, v)$, without considering its "context", i.e., other cookies registered by the same web server.

Known Names. Perhaps the most surprising evidence we got from our investigation is that many cookies which comply with standard naming conventions for authentication cookies are *not* actually authentication cookies. For instance, it is not true that all the cookies called `PHPSESSID` (the standard name adopted by PHP for authentication cookies) are really used for authentication purposes. To conclude this, we matched the

cookie names occurring in our ground truth against an extensive list of 45 known standard names employed by SERENE. The distribution of these cookies in the two classes is shown in Table IX.

Table IX: Distribution of cookies with known authentication cookie names

		Authentication Cookie	
		yes	*no*
$known(k)$	*yes*	55	56
	no	277	2,076

We isolated 111 matching cookies overall: interestingly, only 55 of them are really authentication cookies (49.5%). We do not have any definite explanation about this unexpected behavior, though we occasionally observed that web developers generate random cookies through some session management API of the underlying framework, and then populate other cookies with these randomly generated values to implement a custom authentication scheme. We observe then that 277 out of 332 (83.4%) authentication cookies in our ground truth do not follow standard naming conventions, which confirms that many websites do not adopt existing APIs for session management.

Still, despite the lack of definite standards, we are also able to confirm that custom names of authentication cookies typically contain authentication-related terms. We isolated from our ground truth all the cookies whose name contains any of the following strings: `sess`, `sid`, `auth`, `user`, `token`, `acc`, `password`, `hash`, `login`, `key`, `security`. The distribution of the two classes of cookies over the partition induced by the outcome of the name matching test is shown in Table X.

Table X: Distribution of cookies with names including authentication-related terms

		Authentication Cookie	
		yes	*no*
$auth(k)$	*yes*	194	303
	no	138	1,829

We observe that 194 out of 332 authentication cookies (58.4%) include at least one of the strings above in their name. The occurrence of those strings in the names of non-authentication cookies is much less frequent, since only 303 out of 2,132 non-authentication cookies (14.2%) pass the naming test.

Value Length. We investigated whether the length of a cookie value is useful to single out the authentication cookies. Fig. 4 provides a box-plot of the distribution of the two classes of cookies with respect to the length of their value.

We can confirm that most of the authentication cookies are rather long as expected, in that their values include at least 25 characters, even though we observe that some authentication cookies are surprisingly short. We performed a further investigation on these unexpected cases and we noticed that most of them are confined into specific websites, which rely on authentication tokens of size 2 to track the requesting client. Typically, these authentication tokens contain a (short) user identifier and a (longer) session identifier, which is likely derived from the user's password and some random value to ensure freshness.

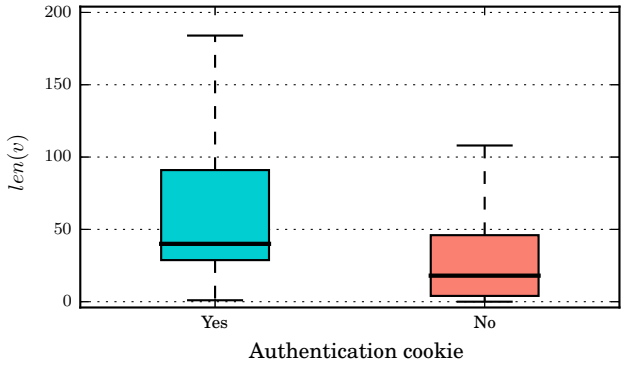

Fig. 4: Distribution of cookies with respect to their value length

Value Randomness. We finally investigated which measure of randomness is more effective at detecting authentication cookies: the password strength s, the Shannon entropy H or the index of coincidence IC. We summarize our findings in Fig. 5. All the three measures look useful for detecting authentication cookies, with the index of coincidence being clearly the most discriminating feature.

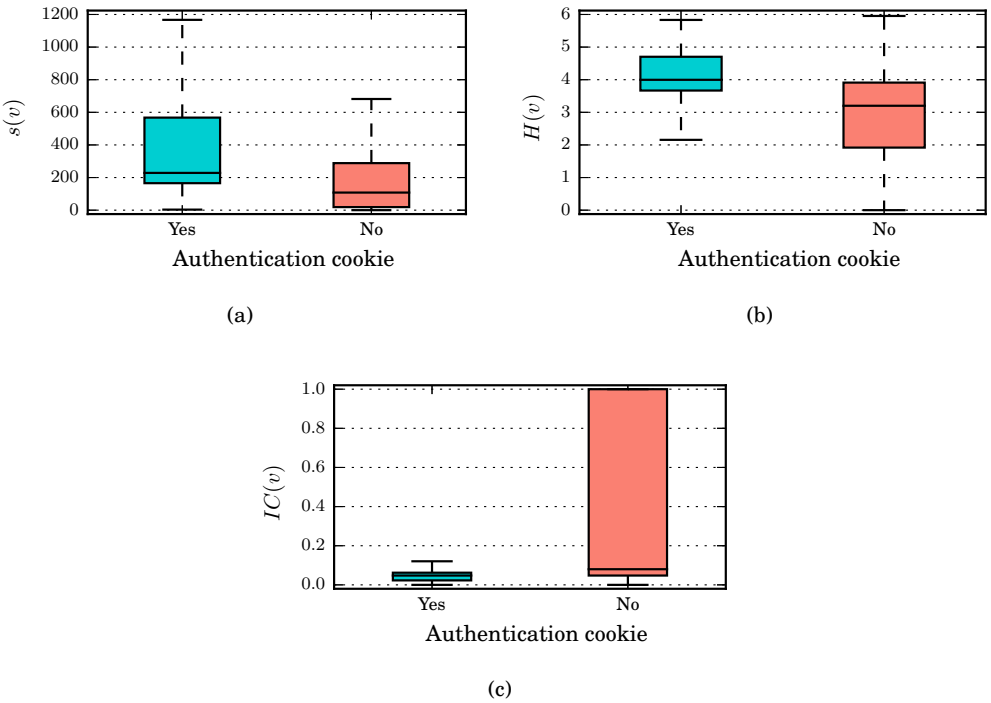

Fig. 5: Distribution of cookies with respect to randomness measures

Expiration Date. A cookie can be assigned by the remote server an expiration date expressed as a Unix timestamp, to instruct the browser to delete it after a given time. Before we started our investigation, we thought that authentication cookies typically had a rather short expiration date, since using the same cookie for a large time window ultimately weakens the session against hijacking. Instead, we noticed that several authentication cookies present a fairly late expiration date, hence this feature is not very effective in practice, as it is highlighted by the box-plot in Fig. 6.

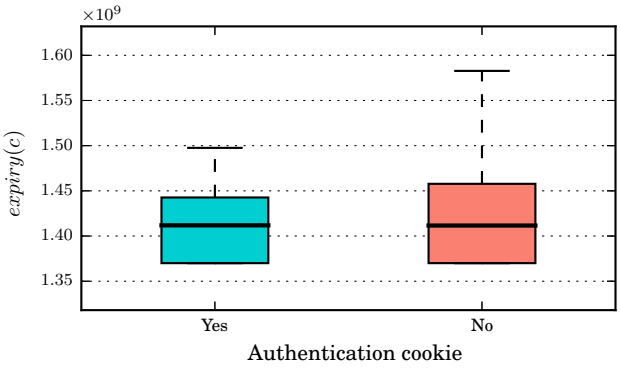

Fig. 6: Distribution of cookies with respect to their expiration date (Unix timestamps)

It seems that different choices correspond to different web application semantics: security-critical websites typically adopt authentication cookies with an early expiration date, while websites with more relaxed security requirements register authentication cookies which expire much later, thus enhancing the experience of users who will need to re-authenticate themselves far less frequently.

5.2.2. Novel Contextual Features. After a careful evaluation of existing proposals, we manually inspected our ground truth, trying to identify novel distinctive features useful for detecting authentication cookies. Our key insight is that the Web is highly *heterogeneous* and that, in general, features which are effective for a given website are not necessarily adequate for another website. We discovered, however, that taking into account all the cookies $\mathcal{C}$ registered by a given website, that is considering the "context" of a given cookie $c \in \mathcal{C}$, we can assign more discriminating features to c.

Term Frequency-Inverse Document Frequency. Assume that some cookies are marked as `HTTP-Only` and/or `Secure`. It would be tempting to conclude that all these cookies are likely used for authentication, since they are explicitly protected by web developers. On the other hand, several studies highlight that the adoption of these cookie flags is largely disregarded by existing websites [Bugliesi et al. 2014a; Zhou and Evans 2010; Nikiforakis et al. 2011]. We show in Table XI the distribution of our two classes of cookies with respect to the presence of the `HTTP-Only` flag, which confirms the previous observation: the flag should likely play a role for classification purposes, but it does not provide a definite evidence of the cookie being used for authentication.

Interestingly, though, a manual investigation of our ground truth reveals that the usage of the cookie flags follows several different patterns, which can (and should) be effectively exploited to single out authentication cookies: first, there are websites which explicitly protect only cookies containing session information; then, we have some high-security websites, e.g., Dropbox, which protect *all* the cookies they register,

Table XI: Distribution of cookies with respect to the `HTTP-Only` flag

		Authentication Cookie	
		yes	*no*
$httponly(c)$	*yes*	143	219
	no	189	1,913

irrespective of the nature of their contents; and finally, we have several websites which just completely ignore the adoption of the available cookie protection mechanisms.

Intuitively, if a website registers a set of cookies, but only one (or very few) of those is labeled as `HTTP-Only`, then that cookie is likely used for authentication purposes. Instead, if all the cookies (or no cookies at all) from the website are labeled as `HTTP-Only`, the presence (or the absence) of the flag should not play any significant role during classification. To catch this behavior, we borrow an idea from the Information Retrieval (IR) field, where the effectiveness of an IR system also depends on an accurate estimate of the importance of each word with respect to the text documents in a given corpus. This estimate is typically captured by the *term frequency-inverse document frequency* (tf-idf) score [Salton and McGill 1986], which increases proportionally to the number of times a word appears in a document, but is penalized by the frequency of the word in the whole corpus.

Specifically, let $\mathcal{C} = \{c_1, \ldots, c_n\}$ be a set of cookies registered by a given website and let $n_h \leq n$ be the number of `HTTP-Only` cookies in $\mathcal{C}$. We define the "term frequency" of the `HTTP-Only` flag for a given cookie c_i as:

$$tf_{\texttt{HTTP-Only}}(c_i) = \begin{cases} 1 & \text{if } c_i \text{ is } \texttt{HTTP-Only} \\ 0 & \text{otherwise.} \end{cases}$$

We then define the "inverse document frequency" of the `HTTP-Only` flag with respect to the set $\mathcal{C}$ as:

$$idf_{\texttt{HTTP-Only}}(\mathcal{C}) = \log_2 \left(\frac{n}{n_h + 1} \right).$$

According to the definition of tf-idf, we then compute:

$$tf\text{-}idf_{\texttt{HTTP-Only}}(c_i, \mathcal{C}) = tf_{\texttt{HTTP-Only}}(c_i) \cdot idf_{\texttt{HTTP-Only}}(\mathcal{C}).$$

In Fig. 7 we show the corresponding box-plot.

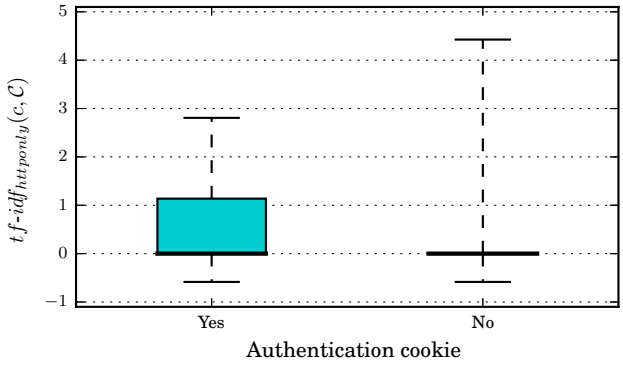

Fig. 7: Distribution of cookies with respect to tf-$idf_{\texttt{HTTP-Only}}$

We observe that the large majority (around 75%) of the non-authentication cookies gathers between -0.5 and 0, while approximately half of the authentication cookies gathers between 0 and 1, hence the feature looks promising for classification.

Z-Score. We observed before that authentication cookies typically contain rather long values. Still, two caveats apply. First, the notion of "long" is inherently dependent on the specific website; indeed, it would be much more accurate to state that authentication cookies are usually longer than any other cookie registered by the *same* website. Second, we occasionally noticed some short authentication cookies that contradict the previous observation. However, we also discussed that these cookies are typically paired with other, longer authentication cookies, and protecting the latter would be enough to safeguard the website. Hence, reasoning at the website level seems effective also to protect scenarios implementing the authentication scheme discussed before, and we thus propose to consider the *Z-score* [Kreyszig 1979] of the cookie length.

In general, given a feature of interest, the Z-score measures the (signed) number of standard deviations an example is above the mean of the population. Concretely, for each $c_i = (k_i, v_i)$ in a set of cookies $\mathcal{C}$ registered by a given website, this is computed as:

$$Z_{length}(c_i, \mathcal{C}) = \frac{length(v_i) - \mu_{\mathcal{C}}}{\sigma_{\mathcal{C}}},$$

where $\mu_{\mathcal{C}}$ and $\sigma_{\mathcal{C}}$ are the *population* mean and standard deviation of the cookie lengths in $\mathcal{C}$, respectively. It is worth noting that we are associating each website to a *separate* random variable, which represents all the possible values of cookie length that are found in $\mathcal{C}$. Therefore, it makes sense assuming to know both $\mu_{\mathcal{C}}$ and $\sigma_{\mathcal{C}}$, since the sample of observed cookies in $\mathcal{C}$ we used to compute their unbiased estimates (i.e., $\hat{\mu}_{\mathcal{C}}$ and $\hat{\sigma}_{\mathcal{C}}$, respectively) is actually the whole population of observable data for that particular website.

We show the box-plot of the Z-score of the value length with respect to the two classes of cookies in Fig. 8. Again, we observe a nice discriminative power of a contextual feature, which suggests its adoption for classification purposes.

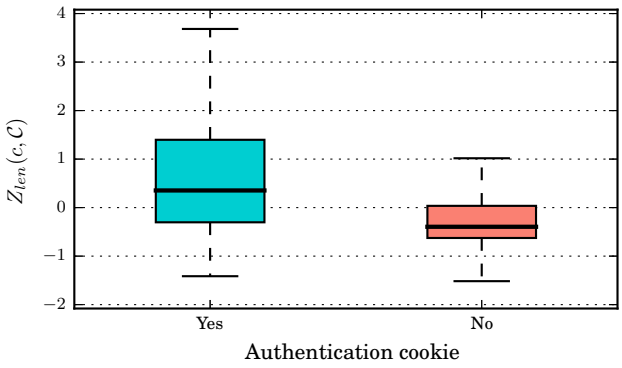

Fig. 8: Distribution of cookies with respect to the Z-score of their length

5.2.3. Feature Selection. The analysis conducted on the previous section gives a good intuition about the features which look most useful for classification. Based on an initial set of apparently promising features, we identified the subset of the 15 most

useful features by using the *feature selection* facility implemented in the scikit-learn[9] Python package. Concretely, we used a forest of trees to derive features importances: each tree provides a relative measure known as "Gini importance", which is defined as the improvement in the "Gini gain" splitting criterion; by averaging these relative importances over several randomized trees, it is possible to reduce the variance of the estimate and use it for feature selection [Breiman 2001; Archer and Kimes 2008].

We show in Fig. 9 the importance of the 15 best features, estimated on our dataset using a forest of 250 trees. The bars show the feature importances estimated by scikit-learn, along with their inter-trees standard deviation. It is worth noting that, the highest the importance score (i.e., *rank*) of a feature the more "powerful" and "effective" that feature is on classifying instances.

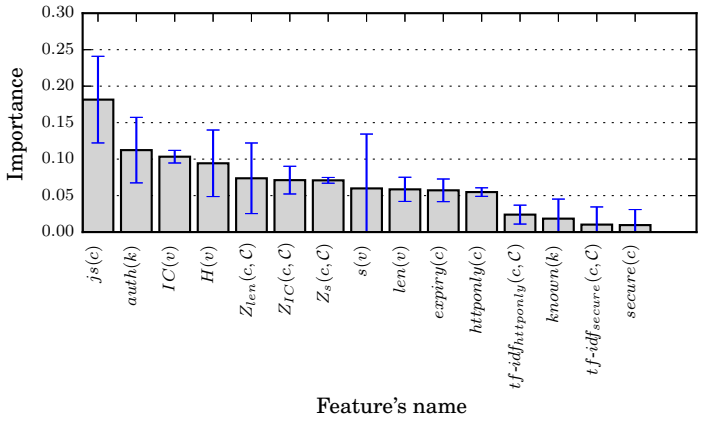

Fig. 9: Feature importance

5.3. Training and Evaluating Classifiers

We trained different machine learning algorithms from the comprehensive collection available in scikit-learn, we applied the learned models to predict the class of cookies in the ground truth, and finally we computed the validity measures introduced in Section 4.1. Since the distribution of the two classes in our ground truth is highly skewed, we have to expect that the plain classifiers trained on this dataset may lead to poor performances, since the predictions are dominated by the most common class (negatives in our applications). This is confirmed by the results reported in Fig. 10. It turns out that only the Bernoulli Naïve Bayes classifier provides acceptable performances, behaving better than the heuristics proposed in the literature so far. However, the sensitivity of the classifier is still too low to guarantee a satisfactory level of protection: we observe that only 216 out of 332 authentication cookies are detected and only the 73% of the websites can be protected. All the other classifiers tend to over-predict the most frequent class: for instance, this is apparent in the confusion matrix of the Random Forest classifier, given in Table XII.

To improve the classifier performances in presence of training sets with skewed class distribution, a common solution is to adopt cost-sensitive learning, illustrated in Appendix A.1. Luckily, this technique significantly improves the performances of all the

[9]http://scikit-learn.org/stable/

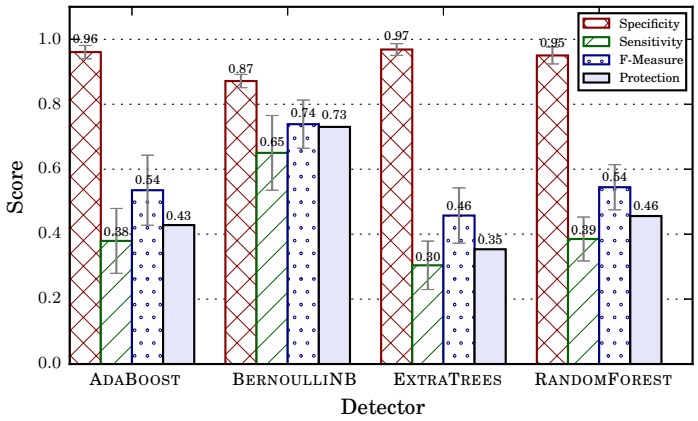

Fig. 10: Evaluation of different classifiers (no cost matrix)

Table XII: Confusion matrix for Random Forest (no cost matrix)

		Predicted	
		positive	*negative*
Actual	*positive*	*tp*: 128	*fn*: 204
	negative	*fp*: 106	*tn*: 2,026

induced classifiers. We summarize the experimental results of the most effective classifiers in Fig. 11.

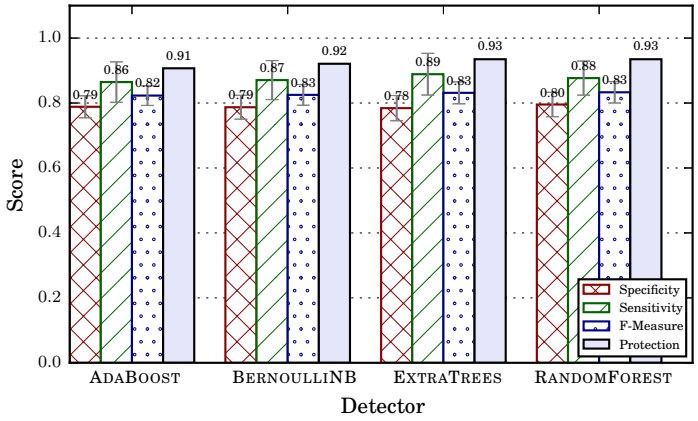

Fig. 11: Evaluation of different classifiers (with cost matrix)

We immediately notice that our machine learning approach outperforms the discussed hand-coded heuristics (see Fig. 3 for comparison), obtaining a remarkable trade-off between sensitivity and specificity, in that the F-measure of the worst classifier (0.82) is significantly higher than the F-measure of the best heuristic (0.67). Overall, we observe that the considered classifiers are able to protect more than 90% of the considered websites, while the best performing heuristics could only safeguard

approximately the 65% of the websites, with the exception of the detector implemented in SESSIONSHIELD, which however has a poor specificity and is not usable in practice. This huge improvement in protection does not come at the cost of usability, since the specificity of each classifier is no worse than the specificity computed for the different heuristics. Only the heuristic adopted in SERENE is slightly better in this respect, but it could only protect around the 40% of the websites. Different trade-offs between security and usability may be chosen by training and testing other classifiers.

For the sake of completeness, we also report in Table XIII the confusion matrix for the Random Forest classifier, assuming that a non-uniform cost matrix is used: note that we are now able to correctly detect 291 out of 332 authentication cookies (88%), whereas we could only detect 128 of these cookies (39%) under the uniform cost model used to build Table XII.

Table XIII: Confusion matrix for Random Forest (with cost matrix)

		Predicted	
		positive	*negative*
Actual	*positive*	tp: 291	fn: 41
	negative	fp: 436	tn: 1,696

6. RELATED WORK

In Section 4 we have already conducted an in-depth discussion of various existing client-side defenses for cookie-based sessions [Bugliesi et al. 2014a; Nikiforakis et al. 2011; De Ryck et al. 2012; Tang et al. 2011]. Browser-based protection mechanisms have also been proposed against other web security threats, most notably CSRF attacks [De Ryck et al. 2010; De Ryck et al. 2011; Johns and Winter 2006]. Simply put, the core idea underlying these solutions is to strip *all* the cookies which would be attached to cross-site HTTP(S) requests: we believe that these defenses could be made less invasive and more usable by deploying them on top of our classifier, to selectively remove from cross-origin requests only the authentication cookies.

Accurately detecting authentication cookies becomes more and more important as the security policy applied by the client-side defense mechanism gets more invasive and complicated. A very recent paper proposes SESSINT, a browser extension aimed at protecting web authentication against a wide range of different threats [Bugliesi et al. 2014b]. The security policy enforced by this extension is significantly more sophisticated than competitor solutions, but the current prototype of SESSINT reuses the simple authentication cookie detector employed by COOKIEXT. We believe that the usability of SESSINT could significantly benefit of the introduction of a more accurate classifier for authentication cookie detection.

Another recent research paper sharing interesting similarities with our approach focuses on the privacy implications of third-party cookies [Roesner et al. 2012]. The authors devise a taxonomy for third-party trackers on the Web, based on their observable behavior, and create a Firefox add-on called TrackingTracker, which automatically classifies web trackers at the client-side according to this taxonomy. The tool is used to collect data about real-word third party trackers and design more effective solutions to improve user's privacy. Notably, however, the classification process does not require machine learning techniques, since it is just accounted for by simple checks on the browser-server interactions.

The security implications of using cookies for authentication purposes have been first studied in a classic paper by Fu et al. [Fu et al. 2001]. The work underlines the cryptographic weaknesses of many cookie schemes deployed in 2001 and provides

guidelines for implementing cookies correctly. Using authentication cookies which are hard to guess and forge is crucial for web session security, but in this paper we address the orthogonal problem of securing cookies which have been properly designed.

To the best of our knowledge, we are the first to leverage machine learning techniques to protect web authentication. Machine learning approaches, however, have been proposed in other areas of computer security, including intrusion detection systems [Sommer and Paxson 2010] and spam filters [Guzella and Caminhas 2009]. In this work, we do not need to consider the challenges of an adversary trying to confuse the machine learning algorithm to avoid detection: websites may not comply with recommended security practices, but they are not deliberately trying to hide which cookies are used for authentication purposes.

7. CONCLUSION

We showed that all the client-side defenses proposed so far to automatically strengthen web authentication are based on heuristics for authentication cookie detection which perform much worse than expected when they are evaluated on a ground truth of authentication cookies. We argue that these simple heuristics are inherently limited, since client authentication on the Web is based on complex, often hard-to-predict usages of authentication cookies. Our conclusion is then that any protection mechanism built on top of these heuristics is bound to either negatively affect the user experience or to provide an unsatisfactory level of protection when deployed on a large scale.

We thus advocated the adoption of supervised learning techniques to automatically detect authentication cookies at the client side: such a machine learning approach builds on powerful, well-established tools to automatically build detectors which outperform current state-of-the art solutions. We experimentally showed a significant improvement in the classification process with respect to existing approaches: our supervised learning solution is very precise, being able to strike a good balance between security and usability. We believe that future client-side defenses for web authentication could significantly benefit from the usage of authentication cookie detectors based on supervised learning techniques.

As a next step, we plan to investigate the generalization of our approach to identify tracking cookies stored in the browser [Roesner et al. 2012]. We would also like to study the definition of a simple contract language for cookie classification purposes, aimed at providing remote servers with an effective way to inform the browsers about the usage of the cookies they register: this would be very important to define and enforce more expressive client-side security policies. Finally, we plan to leverage Amazon Mechanical Turk[10] as an effective tool to extend the construction of our ground truth of cookies: by asking humans to register to other websites of interest, we could reuse our Python script to automatically collect many more authentication cookies.

ACKNOWLEDGMENTS

We would like to thank Nick Nikiforakis, Philippe De Ryck, and Shuo Tang for disclosing full details about the authentication cookie detectors employed by SESSIONSHIELD, SERENE, and ZAN. The anonymous reviewers provided valuable feedback to improve the presentation of the paper.

REFERENCES

Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast algorithms for mining association rules. In *International Conference on Very Large Data Bases (VLDB)*. 487–499.

Kellie J. Archer and Ryan V. Kimes. 2008. Empirical characterization of random forest variable importance measures. *Computational Statistics & Data Analysis* 52, 4 (2008), 2249–2260.

[10]https://www.mturk.com/mturk/welcome

L. Breiman. 2001. Random Forests. *Machine Learning* 45 (2001), 5–32.

Carla E. Brodley and Paul E. Utgoff. 1995. Multivariate Decision Trees. *Machine Learning* 19, 1 (1995), 45–77. http://dblp.uni-trier.de/db/journals/ml/ml19.html#BrodleyU95

Michele Bugliesi, Stefano Calzavara, Riccardo Focardi, and Wilayat Khan. 2014a. Automatic and robust client-side protection for cookie-based sessions. In *Engineering Secure Software and Systems (ESSoS)*. 161–178.

Michele Bugliesi, Stefano Calzavara, Riccardo Focardi, Wilayat Khan, and Mauro Tempesta. 2014b. Provably sound browser-based enforcement of web session integrity. In *IEEE Computer Security Foundations Symposium (CSF)*. To appear.

Stefano Calzavara, Gabriele Tolomei, Michele Bugliesi, and Salvatore Orlando. 2014. Quite a mess in my cookie jar!: leveraging machine learning to protect web authentication. In *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014*. 189–200.

Nitesh V. Chawla. 2005. Data Mining for Imbalanced Datasets: an Overview. (2005).

Italo Dacosta, Saurabh Chakradeo, Mustaque Ahamad, and Patrick Traynor. 2012. One-time cookies: Preventing session hijacking attacks with stateless authentication tokens. *ACM Transactions on Internet Technology* 12, 1 (2012), 1.

Philippe De Ryck, Lieven Desmet, Thomas Heyman, Frank Piessens, and Wouter Joosen. 2010. CsFire: Transparent Client-Side Mitigation of Malicious Cross-Domain Requests. In *Engineering Secure Software and Systems (ESSoS)*. 18–34.

Philippe De Ryck, Lieven Desmet, Wouter Joosen, and Frank Piessens. 2011. Automatic and Precise Client-Side Protection against CSRF Attacks. In *European Symposium on Research in Computer Security (ESORICS)*. 100–116.

Philippe De Ryck, Nick Nikiforakis, Lieven Desmet, Frank Piessens, and Wouter Joosen. 2012. Serene: Self-Reliant Client-Side Protection against Session Fixation. In *Distributed Applications and Interoperable Systems (DAIS)*. 59–72.

P.A. Devyver and J. Kittler. 1982. *Pattern Recognition: A Statistical Approach*. Prentice-Hall.

Michael Dietz, Alexei Czeskis, Dirk Balfanz, and Dan S. Wallach. 2012. Origin-Bound Certificates: A Fresh Approach to Strong Client Authentication for the Web. In *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*. 317–331.

Charles Elkan. 2001. The Foundations of Cost-Sensitive Learning. In *In Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*. 973–978.

Dinei A. F. Florêncio and Cormac Herley. 2007. A large-scale study of web password habits. In *International Conference on World Wide Web (WWW)*. 657–666.

Seth Fogie, Jeremiah Grossman, Robert Hansen, Anton Rager, and Petko D. Petkov. 2007. *XSS Attacks: Cross Site Scripting Exploits and Defense*. Syngress Publishing.

William F. Friedman. 1922. *The index of coincidence and its applications to cryptanalysis*. Cryptographic Series.

Kevin Fu, Emil Sit, Kendra Smith, and Nick Feamster. 2001. The Dos and Don'ts of Client Authentication on the Web. In *10th USENIX Security Symposium, August 13-17, 2001, Washington, D.C., USA*.

Stuart Geman, Elie Bienenstock, and René Doursat. 1992. Neural networks and the bias/variance dilemma. *Neural Computation* 4, 1 (January 1992), 1–58.

Thiago S. Guzella and Walmir M. Caminhas. 2009. A review of machine learning approaches to Spam filtering. *Expert Systems with Applications* 36, 7 (2009), 10206–10222.

Jeff Hodges, Collin Jackson, and Adam Barth. 2012. HTTP Strict Transport Security. Available online at `http://tools.ietf.org/html/rfc6797`. (2012).

Collin Jackson and Adam Barth. 2008. ForceHTTPS: protecting high-security web sites from network attacks. In *International Conference on World Wide Web (WWW)*. 525–534.

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2014. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated.

Nathalie Japkowicz and Shaju Stephen. 2002. The class imbalance problem: A systematic study. *Intelligent Data Analysis* 6, 5 (2002), 429–449.

Martin Johns, Bastian Braun, Michael Schrank, and Joachim Posegga. 2011. Reliable protection against session fixation attacks. In *ACM Symposium on Applied Computing (SAC)*. 1531–1537.

Martin Johns and Justus Winter. 2006. RequestRodeo: client side protection against session riding. *Proceedings of the OWASP Europe Conference* (2006), 5–17.

Michal Kranch and Joseph Bonneau. 2015. Upgrading HTTPS in mid-air: an empirical study of strict transport security and key pinning. In *Network and Distributed System Symposium (NDSS)*. To appear.

E. Kreyszig. 1979. *Advanced Engineering Mathematics* (4 ed.). Wiley.

Thomas M. Mitchell. 1997. *Machine Learning* (1 ed.). McGraw-Hill, Inc., New York, NY, USA.

Nick Nikiforakis, Wannes Meert, Yves Younan, Martin Johns, and Wouter Joosen. 2011. SessionShield: Lightweight Protection against Session Hijacking. In *Engineering Secure Software and Systems (ESSoS)*. 87–100.

Claudia Perlich, Foster Provost, and Jeffrey S. Simonoff. 2003. Tree induction vs. logistic regression: a learning-curve analysis. *Journal of Machine Learning Research* 4 (December 2003), 211–255.

M. M. Rahman and D. N. Davis. 2013. Addressing the class imbalance problem in medical datasets. *International Journal of Machine Learning and Computing* 3, 3 (2013), 224–228.

Franziska Roesner, Tadayoshi Kohno, and David Wetherall. 2012. Detecting and defending against third-party tracking on the web. In *USENIX Conference on Networked Systems Design and Implementation (NSDI)*. 1–14. http://dl.acm.org/citation.cfm?id=2228298.2228315

G. Salton and M. J. McGill. 1986. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA.

Claude Shannon. 1948. A Mathematical Theory of Communication. *The Bell System Technical Journal* 27 (1948), 379–423. http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf

Robin Sommer and Vern Paxson. 2010. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *IEEE Symposium on Security and Privacy*. 305–316.

Shuo Tang, Nathan Dautenhahn, and Samuel T. King. 2011. Fortifying web-based applications automatically. In *ACM Conference on Computer and Communications Security (CCS)*. 615–626.

Gary M. Weiss, Kate McCarthy, and Bibi Zabar. 2007. Cost-Sensitive Learning vs. Sampling: Which is Best for Handling Unbalanced Classes with Unequal Error Costs? (2007).

Yuchen Zhou and David Evans. 2010. Why Aren't HTTP-Only Cookies More Widely Deployed. In *Web 2.0 Security and Privacy Workshop (W2SP'10)*.

APPENDIX

A.1. Challenges of a Supervised Learning Approach

In the following, we discuss the main aspects which have to be carefully considered when designing any supervised learning solution, with a special emphasis on the choices we applied to our specific setting. We use the same notation introduced in Section 5.1 above.

A.1.1. Choosing the Best Features. A key aspect which deserves attention concerns the choice of the feature space X used to represent instances: indeed, the selected features are crucial for any classifier to be effective. Intuitively, we want to identify highly distinctive properties of the cookies, which would allow the learned classifier to correctly discriminate between instances belonging to different classes. To tackle this problem, we first perform a preliminary manual investigation of a subset of our ground truth and we identify apparently promising features; then, we compute some basic statistics on each of those features with respect to the two classes of cookies from the entire dataset, to get sense of how much discriminant each feature is.

Based on this preliminary investigation, we identify an initial set of apparently useful features and we then apply an automatic *feature selection* process to identify the subset of the features which most positively affect the learning of our classifier. Reducing the set of features is important for three reasons: first, it makes the classification process faster; second, it helps avoiding the risk of *overfitting* the training set (see below); and third, it provides a better understanding of the classification problem. We report the details and the results of this process in Section 5.2.

A.1.2. Bias-Variance Tradeoff. From a formal perspective, finding a good estimate $\lambda \circ \chi$ of the unknown labeling function ℓ is actually an optimization problem which usually requires to minimize a *cost function*. In most cases, the cost function to minimize is the *training error rate* (or *in-sample error rate*), namely the rate of mistakes the classifier does when labeling instances of the training set $\mathcal{D}_{\text{train}}$. A misclassification occurs whenever the actual class label of a cookie c is different from the class label predicted

by the classifier, i.e., whenever $\ell(c) \neq \lambda(\chi(c))$. In practice, however, we would like to choose the classifier with the smallest *test error rate* (or *out-of-sample error rate*), which measures the misclassification rate for instances that are *not* part of the training set.

Unfortunately, a classifier with the smallest training error rate does not necessarily turn into one with the smallest test error rate: this issue is also known as the *bias-variance* tradeoff [Geman et al. 1992]. Roughly speaking, we desire the ability of a learning algorithm to perform effectively on the training set (i.e., preventing *underfitting*), while being flexible enough to fit other datasets than the one on which it is trained (i.e., avoiding *overfitting*).

An immediate way to tackle this problem is to use the so-called *hold-out* method. The idea is to randomly split the ground truth $\mathcal{G}$ into two separate datasets: $\mathcal{G}_{\text{train}} \subset \mathcal{G}$ and $\mathcal{G}_{\text{test}} = \mathcal{G} \setminus \mathcal{G}_{\text{train}}$. The former is used to build the training set $\mathcal{D}_{\text{train}}$ for learning, i.e., to compute an estimate of ℓ as $\lambda \circ \chi$, whereas the latter is used to build the *test set* $\mathcal{D}_{\text{test}}$, employed to measure the test error rate for the found estimate $\lambda \circ \chi$. Though conceptually simple and easy to implement, this approach is very sensitive to the choice of the specific splitting and tends to overestimate the true test error rate. To obtain a less biased estimate, k-fold *cross-validation* is adopted [Devyver and Kittler 1982].

The idea behind k-fold cross-validation is to use the whole ground truth of observations to derive the initial training set $\mathcal{D}_{\text{train}}$, so that $\mathcal{G}_{\text{train}} = \mathcal{G}$. Then, $\mathcal{D}_{\text{train}}$ is randomly partitioned into k equally-sized subsets (folds): $k-1$ subsets are used as training data to learn a classification model and the remaining subset is retained as the hold-out data to assess the model as described above. The cross-validation process is repeated k times, with each of the k subsets used exactly once as the test set: the k validation errors so obtained are then averaged to produce a single estimate of the test error rate. In this paper, we use a *stratified* 10-fold cross-validation strategy, which ensures the same distribution of classes in each fold as in the original dataset. This helps getting more reliable error estimates at each stage of cross-validation, and overall a better estimate of the test error, especially when instances are unevenly distributed on the classes like in our case. For a complete discussion on the cross-validation approach, we refer the reader to [James et al. 2014].

A.1.3. Dealing with Skewed Classes. A preliminary investigation of our ground truth immediately reveals that the distribution of cookies on the two classes of interest is skewed: approximately, there are 7 non-authentication cookies for each authentication cookie. Unfortunately, it is well-known that a classifier learned from a training set which shows a strongly skewed class distribution may lead to poor and misleading performances [Japkowicz and Stephen 2002]. This is caused by the fact that, as stated above, most supervised learning algorithms operate by minimizing the misclassification rate on the training set, hence they typically tend to predict the most frequent class. In order to deal with this issue, a *cost model* is usually introduced [Elkan 2001].

Inspired by the well-known confusion matrix introduced in Section 4 to assess the performance of a binary classifier, cost-sensitive learning approaches introduce a *cost matrix*, so that a classifier is *penalized* when it incorrectly classifies an instance. The cost matrix provides the costs associated with each of the four possible outcomes shown in the confusion matrix, which we denote by $\gamma_{tp}, \gamma_{fp}, \gamma_{tn}, \gamma_{fn}$. In the typical scenario, no costs are assigned to correct predictions, i.e., $\gamma_{tp} = \gamma_{tn} = 0$, while $\gamma_{fp}, \gamma_{fn} > 0$. If $\gamma_{fp} \neq \gamma_{fn}$, the model adopts a *non-uniform* cost model, where $\gamma_{fn} > \gamma_{fp}$ or $\gamma_{fp} > \gamma_{fn}$ on the basis of the type of error we want to penalize the most. Usually, the cost of misclassifying an instance of the rarest class is larger than the cost of misclassifying an instance of the most frequent class (i.e., $\gamma_{fn} > \gamma_{fp}$). More specifically, assuming that p and n are the number of positive and negative instances in the training set, respectively, the cost matrix adopted in the present paper is shown in Table XIV.

Table XIV: Cost matrix

		Predicted	
		positive	*negative*
Actual	*positive*	$\gamma_{tp} : 0$	$\gamma_{fn} : \frac{min(p,n)}{p}$
	negative	$\gamma_{fp} : \frac{min(p,n)}{n}$	$\gamma_{tn} : 0$

Since $p \ll n$ (332 vs. 2,132), it turns out that $\gamma_{fn} > \gamma_{fp}$, which means that our cost model penalizes more the errors affecting instances of the positive class.

Given a cost matrix, cost-sensitive learning may be implemented in two ways. The first strategy, known as *sampling*, factors out the cost model by altering the original training set, so that the initial class distribution is altered proportionally to the (asymmetric) penalties the classifier should pay if a misclassification error occurs. Two basic sampling methods can be used: *oversampling* and *undersampling*. In the former (resp. latter), instances of the less (resp. more) frequent class are replicated in (resp. discarded from) the training set to make it more balanced. The second viable cost-sensitive learning strategy, instead, suggests to plug the cost model directly into the learning algorithm, thus tweaking the cost function to be optimized upon learning.

Choosing the best cost-sensitive learning strategy is still subject of debate [Chawla 2005; Weiss et al. 2007; Rahman and Davis 2013]. There are known disadvantages associated with the use of sampling to implement cost-sensitive learning. The drawback with undersampling is that it discards potentially useful data, whereas with oversampling the risk of overfitting increases, since it replicates exact copies of existing instances. We thus decide to directly include the cost model into the learning process: this restricts the choice of the possible classifiers, since not every classifier supports cost-sensitive learning, but still we are able to identify good performing classifiers for our problem. In Section 5.3 we experimentally confirm that the skewness of class distribution does negatively impact on the classification process and we prove that our choice of the cost matrix is appropriate to deal with this issue.