Quite a Mess in My Cookie Jar!

Leveraging Machine Learning to Protect Web Authentication

Stefano Calzavara calzavara@dais.unive.it
Gabriele Tolomei gabriele.tolomei@unive.it
Michele Bugliesi bugliesi@unive.it
Salvatore Orlando orlando@unive.it

Università Ca' Foscari Venezia, Italy

ABSTRACT

Browser-based defenses have recently been advocated as an effective mechanism to protect 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 gold set of cookies we collect from 70 popular websites of the Alexa ranking. To obtain the gold set, we devise a semi-automatic procedure that draws on a 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 gold set, 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 gold set is used to train a binary classifier, and report on experimental evidence that our method outperforms existing proposals. Interestingly, the resulting classification, together with our hands-on experience in the construction of the gold set, provides new insight on how web authentication is implemented in practice.

Categories and Subject Descriptors

H.4.3 [**Information Systems Applications**]: Communications Applications; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Design, Experimentation, Security

Keywords

Web security, Authentication cookies, Classification

WWW'14, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2744-2/14/04.
http://dx.doi.org/10.1145/2566486.2568047.

1. INTRODUCTION

Both HTTP and its secure variant HTTPS, the workhorse protocols of the current World Wide Web, are stateless by design, hence require web servers to implement their own authentication mechanisms to keep track of state information across different HTTP(S) 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 operating with them involves highly structured interactions (*sessions*), often requesting users to present their credentials 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 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 allows 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 surface of attack against authentication cookies, requiring web servers to deploy a variety of counter-measures to achieve a satisfactory degree of protection. Unfortunately, as reported in the literature, websites often fail to implement such measures correctly [30, 18].

A complementary line of defense, advocated in a series of recent papers, may be built directly within the browser through client-side protection mechanisms [28, 24, 23, 6]. 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. This process ultimately hinges on an *authentication cookie detector*, i.e., a heuristics which tries to automatically identify the cookies associated with the user credentials among all

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

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 the real set of authentication cookies, since the conservative security policy applied for them may negatively impact on the user experience. For example, if a security policy aimed at thwarting XSS attacks stipulates that authentication cookies should not be accessed through JavaScript, then any legitimate access to a cookie storing the user's preferences will be forbidden.

Somewhat surprisingly, in spite of their fundamental impact on 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 quality assessment.

Contributions.

Our first contribution is the design of a (semi-)automatic method to build a gold set 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 derived from a sample of 70 amongst the today's most popular websites of the Alexa ranking, which we make available for public download[2]. Interestingly, our experience in the construction of the gold set has unveiled a number of subtleties in the actual role that different cookies play in web authentication, which appear to have largely been overlooked in the past. Based on that experience, we devise a new notion of *authentication token*, which captures multiple web authentication schemes and nicely fits different real-world situations.

Our second contribution is a rigorous evaluation of four existing authentication cookie detectors, conducted against the gold set 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. By a manual inspection of our gold set, 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 *binary classifier* 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.

The rest of the paper is organized as follows. Section 2 provides background about cookie-based session security. Section 3 describes the gold set 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] `http://bit.ly/GAQluz`

2. BACKGROUND: SESSION SECURITY

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, whereby a script crafted by the attacker runs in the security context of a trusted website [9]. The script is thus allowed to read the authentication cookie value and disclose it to the attacker, thus allowing him to hijack the user's session.

Since XSS attacks are widespread today, 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 registered them. A few research works suggest to automatically apply the `HTTP-Only` flag to authentication cookies when the remote server fails to protect them [28, 18, 6].

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, authentication cookies are leaked over HTTP to any attacker who is able to eavesdrop the unencrypted web traffic [13].

The `Secure` flag can be used by a web server to designate a cookie that should only be sent over HTTPS connections and never be attached to HTTP requests. 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 [6].

2.3 Session Fixation

In a session fixation attack, the attacker is able to designate (e.g., through a script) the value of the authentication cookie which will identify the user's session [14]. The attacker can then impersonate the user after she has performed the required authentication steps, e.g., by submitting her credentials to the website. Notice that in a session fixation the authentication cookie is never leaked to the attacker (the attacker knows it in advance), hence the previous protection mechanisms are clearly bound to fail.

A typical server-side solution against session fixation attacks is to require the generation of a fresh authentication cookie whenever the privilege level of the session changes, e.g., after the user has authenticated to the website: since the new cookie will differ from the fixated one, the attacker will not be able to hijack the session. If the server does not implement this simple recommended 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 [24].

3. A GOLD SET OF COOKIES

Building a gold set of cookies 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, which requires one to possess personal accounts on the websites of interest, and authenticate to their private areas [28]. 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 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, 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 over 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:*

(i) authentication: *S authenticates B for any request including all the cookies in A;*

(ii) minimality: *S does not authenticate B for any request including only cookies in $A' \subset A$.*

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 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.

The notion of authentication token is also useful to evaluate the robustness of a given web authentication scheme.

Definition 3.2 (Vulnerability). *An authentication token A is* vulnerable *if and only if every cookie $c \in A$ is known to the attacker.*

Accordingly, 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 any session hijacking attack.

3.2 Building the Gold Set

Let $\mathcal{C} = \{c_1, \ldots c_n\}$ be the set of all the cookies that server S sends to browser B upon login. To construct our gold set, we need to identify a *labeling function* $l : \mathcal{C} \longmapsto \{0, 1\}$ such that $l(c_i) = 1$ iff c_i is an authentication cookie. Unfortunately, discovering this function 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 tokens can occur in $\mathcal{C}$, the search space we have to consider to derive the labeling function l is the *powerset* of $\mathcal{C}$, whose cardinality is 2^n, i.e., exponential in the number of cookies.

Still, we can significantly improve the efficiency of the gold set construction by designing a smarter visit and pruning strategy 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 [3]: if an itemset I is frequent in a transactional database, then any $I' \subset I$ is frequent as well. The *Apriori* algorithm [3] for mining frequent itemsets exploits this property to reduce its exponential search space, by exploring the candidate itemsets from the smallest to the largest.

Similarly to Apriori, our solution (cf. Algorithm 1) iteratively generates and checks subsets of cookies of size k, for $k = [1, n]$. We enumerate all the subsets of cookies from the smallest to the largest, and as soon as we find a subset that authenticates, that is surely an authentication token to be returned, due to the minimality property of the tokens. The set $cand^{(k)}$ denotes the collection of candidate cookies of size k built by the algorithm. To generate $cand^{(k)}$, $k > 1$, function `Gen&Prune` (line 12) exploits the anti-monotonicity property as follows:

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

where $|C^{(k)}| = k$ and $\mathcal{A}$ is the set of authentication tokens mined so far (of size up to $k-1$). For each $C^{(k)} \in cand^{(k)}$, function `isAuthenticated` (line 7) checks whether $C^{(k)}$ authenticates B at S (see Section 3.3). Note that, by construction of $cand^{(k)}$, given $C^{(k)} \in cand^{(k)}$, for all the subsets of $C^{(k)}$ server S does not authenticate browser B. So, if $C^{(k)}$ allows B to be authenticated by S, it is surely an authentication token due to minimality, and thus can be added to the set $\mathcal{A}$ of authentication tokens (line 8). The algorithm stops when no candidate of size k is available.

Like Apriori, the worst-case time complexity of the algorithm is still exponential in the number of cookies. Specifically, this happens when the entire set of cookies $\mathcal{C}$ is the only authentication token, and thus no candidate pruning can occur. In our experiments we verified that this is very

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      A ⟵ ∅;
3      k ⟵ 1;
4      cand^(k) ⟵ {{c} | c ∈ C};
5      while cand^(k) ≠ ∅ do
6          foreach C^(k) ∈ cand^(k) do
7              if isAuthenticated(B, S, C^(k)) then
8                  A ⟵ A ∪ {C^(k)};
9              end
10         end
11         k ⟵ k + 1;
12         cand^(k) ⟵ Gen&Prune(k, C, A);
13     end
14     return A;
15 end
```

unlikely, making the approach generally effective in guaranteeing a significant reduction of the search space.

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 gold set of cookies for S, we can finally derive the labeling function $l : \mathcal{C} \longmapsto \{0, 1\}$, where $l(c) = 1$ 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 gold set $\mathcal{G}$:

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

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

3.3 Implementation and Assessment

We implemented our gold set construction algorithm in Python, using the Mechanize library for stateful programmatic web browsing [1]. 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.

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 either of the following two conditions is true:

1. the response from w does not contain a login form;
2. the response from w contains the username u.

The rationale of this choice is based on the common practice implemented by existing websites, which typically display a login form when the user logs out, and/or include in their pages the username associated with the ongoing session.

We used our script to crawl 70 popular websites from the Alexa ranking, collecting a dataset of 327 cookies, including 103 authentication cookies. We observe that 52 websites (74.3%) only use one authentication token, while the remaining 18 (25.7%) register two different tokens. We also notice that a non-negligible fraction of the considered websites (20.0%) employ authentication tokens of size larger than 1.

A few remarks are in order on our gold set. First, even though we automated the construction, clearly one still must possess a personal account on the considered websites, for which the initial registration is inherently manual. Second, the percentage of authentication cookies (and the number of cookies itself) may look surprising when contrasted with the experience of ordinary web surfing. In particular, browsers typically store a far larger number of cookies per website, but only relatively few of them are used for authentication. This gap is readily explained, since the Mechanize library does not interpret JavaScript, which is a significant source of the cookies generated for each website. We believe (as per previous studies [24, 28]) that authentication cookies are very rarely set via JavaScript.

To check the correctness of our implementation and validate the gold set, we accessed all the considered websites from a browser and we manually 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, providing incorrect answers only for 2 of the 70 websites: both failures were due to websites deviating from the expected behavior assumed by our implementation of the `isAuthenticated` check, and they were later fixed manually.

4. EVALUATING CLIENT-SIDE DEFENSES

In this section we assess the quality of state-of-the-art authentication cookie detectors against our gold set. As we report below, previous evaluations turn out to be way too optimistic, leaving large room for improvement in the detection process.

4.1 Existing Solutions

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

- SessionShield [18] is a proxy between the browser and the network which aims at protecting cookie-based sessions against XSS attacks by emulating the browser behavior implemented for the `HTTP-Only` flag;
- Serene [24] 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 [6] is an extension for Google Chrome designed to ensure the confidentiality of authentication cookies against both XSS attacks and 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 [28] is an extension for the OP2 web browser aimed at protecting legacy web applications against different kinds of vulnerabilities. Notably, ZAN tries to automatically apply the `HTTP-Only` flag to authentication cookies, to prevent their leakage via XSS.

All the authentication cookie detectors adopted by these tools are based on a set of *hand-coded* rules, which rely 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 chosen for the same feature), we abstract from the details here, and just summarize the aspects that are most relevant to our present needs. For any further information, we refer to the original papers.

Table 1 provides an intuitive description of the checks performed by each detector. Cookies are denoted by pairs $c = (k, v)$, and each detector is represented at the bottom of the table as a boolean formula $\phi(c)$, which holds true if and only if c is recognized as an authentication cookie.

4.2 (Re-)Evaluating Existing Detectors

4.2.1 Validity Measures

We implemented each of the four detectors in Table 1, and we fed them with all the cookies in our gold set, to then count 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 gold set 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 gold set.

For each detector, we then computed two standard measures aimed at estimating its effectiveness:

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

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. Instead, 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.

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.

4.2.2 Criticisms to Previous Assessments

The assesment of existing authentication cookie detectors has so far been organized around (*i*) the collection of a dataset of cookies 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 of 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 [17]. 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.

4.2.3 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 2.

		Predicted	
		positive	*negative*
Actual	*positive*	*tp*: 95	*fn*: 8
	negative	*fp*: 105	*tn*: 119

Table 2: Confusion matrix for SessionShield

We notice that sensitivity is impressive (92%), since the algorithm produces only 8 *fn*. On the other hand, specificity is very low (53%), as 105 *fp* are produced by the detector on our set of 327 cookies. This strongly conflicts with the preliminary evaluation performed by the authors of SESSIONSHIELD, who estimated only 19 *fp* in a set of 2167 collected cookies: the difference between the informal estimation above and the formal evaluation we conduct 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 more accurate solution to deal with several usability issues, due to a rather high number of *fp* identified in practice [24]. Indeed, the detection algorithm of SESSIONSHIELD appears too inaccurate to live up to any large-scale usability study of a client-side defense based on it.

Check	Description	Shield	Serene	CookiExt	Zan
$auth(k)$	k contains authentication-related terms (e.g., "*sess*")	✔	✔	✔	✔
$known(k)$	k is a standard session identifier name (e.g., `PHPSESSID`)		✔		
$len(v)$	the length of v is above a given threshold	✔	✔	✔	✔
$H(v)$	the Shannon entropy [26] of v is above a given threshold				✔
$IC(v)$	the index of coincidence [10] of v is below a given threshold			✔	
$\rho(v)$	the password strength [8] of v is above a given threshold	✔	✔		
$dict(v)$	v matches a dictionary word	✔	✔		

$\textsc{Shield}(c) \triangleq (auth(k) \wedge len(v)) \vee (\rho(v) \vee \neg dict(v))$
$\textsc{Serene}(c) \triangleq known(k) \vee (auth(k) \wedge \rho(v) \wedge \neg dict(v) \wedge len(v))$
$\textsc{CookiExt}(c) \triangleq auth(k) \vee (IC(v) \wedge len(v))$
$\textsc{Zan}(c) \triangleq (auth(k) \wedge (H(v) \vee len(v))) \vee (\neg auth(k) \wedge H(v) \wedge len(v))$

Table 1: Checks performed to label the cookie $c = (k, v)$ as an authentication cookie

Next, we turn to Serene, whose confusion matrix is given in Table 3 below.

		Predicted	
		positive	*negative*
Actual	*positive*	*tp*: 48	*fn*: 55
	negative	*fp*: 37	*tn*: 187

Table 3: Confusion matrix for Serene

The number of *fp* decreases significantly (37 vs. 105) with respect to the authentication cookie detector of SessionShield and the specificity greatly improves (83%). On the other hand, we notice that only 48 out of 103 authentication cookies are successfully recognized, and the sensitivity of the detector is really low (47%). We observe that the 55 *fn* are distributed on 44 different websites: if we assume that all these cookies can be fixated by an attacker, we get 40 *vulnerable* authentication tokens (cf. Definition 3.2) on 40 different websites. Hence, Serene is effective at protecting only 30 out of 70 websites (42.9%), which is far less than the empirical estimate in the original paper (83.4%).

To be fair, we also evaluate CookiExt, a solution proposed by two of the authors. Similarly to what happened for the other proposals, our initial evaluation of CookiExt provided some overly optimistic results. We report in Table 4 the result of our later assessment against the gold set.

		Predicted	
		positive	*negative*
Actual	*positive*	*tp*: 68	*fn*: 35
	negative	*fp*: 60	*tn*: 164

Table 4: Confusion matrix for CookiExt

The protection offered by CookiExt appears significantly stronger than the previous one, since the number of *fn* produced by the detector is much lower (35 vs. 55) and sensitivity increases (66%). Still, the degree of protection is lower than our original estimate, and certainly needs improvement: indeed, our 35 *fn* occur in a set of 327 cookies, while the informal estimate in the CookiExt paper identifies only 29 potential *fn* in a much larger set of 2291 collected cookies. Again, the difference is of one order of magnitude and the impact on security is significant.

We conclude with Zan, whose confusion matrix is given in Table 5.

		Predicted	
		positive	*negative*
Actual	*positive*	*tp*: 58	*fn*: 45
	negative	*fp*: 37	*tn*: 187

Table 5: Confusion matrix for Zan

Again, we have a considerable number of *fn*, since 45 out of 103 authentication cookies are misclassified, and the sensitivity is unsatisfactory (56%). However, the original paper on Zan does not consider the accuracy of the authentication cookie detector itself, rather it provides an evaluation about the gain in protection enabled by the tool through an analysis at the website level. In particular, the authors state that 103 out of 136 considered websites (75.7%) are successfully protected by Zan: all the 33 failures are due to the underlying detection algorithm. To carry out a fair comparison, we also focus on the websites rather than on the cookies: we perform a manual review of the 45 *fn* above and we identify 29 vulnerable authentication tokens in 29 different websites. Hence, we conclude that only 41 out of 70 considered websites (58.6%) are actually protected.

To conclude, we argue that the only authentication cookie detector which presents a satisfactory sensitivity is the one employed by SessionShield. Unfortunately, that detector has very low specificity and is thus not fit for any practical client-side defense. In all the other cases, instead, our evaluation highlights an unexpectedly low degree of protection.

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 through manual inspection: this is exactly one of the reasons why previous informal evaluations turned out to be imprecise and ultimately unreliable.

5.1 A Supervised Learning Approach

We automatically build our detector by learning a *binary classifier* from our trusted gold set of cookies. More specifically, for each cookie c in our gold set $\mathcal{G}$ we need to identify an M-dimensional vector of *features* $\mathbf{v}(c) = (f_1, \ldots, f_M)$, where each feature f_i describes a particular aspect of the cookie. Eventually, a tuple $(\mathbf{v}(c), l(c))$ is associated with each $c \in \mathcal{G}$.

To properly learn a binary classifier, we need a *training set*, i.e., a subset of the gold set containing both positive and negative examples, which is used to infer a classification model. The remaining labeled examples of the gold set are used as *test set*, and are exploited to evaluate the performance of the trained model. Finally, the binary classifier derived from the training set is a function that maps the feature vector $\mathbf{v}(c')$ of an *unknown* cookie c' to either a positive or negative class label. The evaluation of the classifier is based on the corresponding confusion matrix.

Besides the availability of a reliable gold set, two main issues have to be carefully addressed when designing any supervised learning solution: the set of features to extract from each object and the actual learning algorithm used for inducing the classifier. Indeed, the features selected are crucial for any classifier to be effective. Intuitively, the goal is to identify highly distinctive object properties, which allow the learned function to correctly discriminate between instances belonging to one of the two classes. Moreover, there exist several learning algorithms, each of which may perform well or poorly, depending on the specific application task [5, 19]. Surely, there are many other aspects of the learning problem, which would deserve to be considered. In this paper we limit our discussion to the fundamental concepts above, and we refer to [29, 17] for any further detail.

5.2 Exploring Cookie 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, i.e., 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 gold set.

5.2.1 Non-contextual Features

We first started by exploring the effectiveness of some features illustrated in Table 1, and exploited by authentication cookie detectors already 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(k).* 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 [2]) are really used for authentication purposes. To conclude this, we matched the cookie names occurring in our gold set against an extensive list of 45 known standard names employed by Serene. We isolated 37 matching cookies overall: interestingly, only 21 of them were really authentication cookies. 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. Notice that 82 out of 103 authentication cookies in our gold set do not comply with standard naming conventions, which confirms that many websites do not directly adopt existing APIs for session management.

– *len(v).* We investigated whether the length of a cookie value is useful to single out the authentication cookies. Figure 1(a) plots the distribution of the two classes of cookies with respect to the length of their value (for presentation purposes, we include in the plot only cookies whose value length is at most 50, which approximately amount to the 80% of our gold set). 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.

– $\rho(v)$, $H(v)$, $IC(v)$. We finally investigated which measure of randomness is more effective at detecting authentication cookies. Here, the most interesting finding is related to the password strength measure ρ adopted by SessionShield. Let $\rho(v) = \mathit{length}(v) \cdot \log_2(\mathit{alphabet}(v))$, where $\mathit{alphabet}(v)$ is a score based on the occurrence in v of lower case letters, upper case letters, digits, and punctuation characters [8]. We show the distribution of the two classes of cookies with respect to the password strength of their value in Figure 1(b) (for presentation purposes, we consider in the plot only cookies whose value length is at least 25, which include about the 86% of all the authentication cookies in our gold set). The observation is that, though fairly long, several authentication cookies are still penalized by the password strength measure, since they do not draw upon a large alphabet of different symbols. Specifically, 53 out of 103 authentication cookies only use one case of letters and digits. It is interesting to notice also that, if we focus on the 84 non-authentication cookies whose value contains at least 25 characters, we find 54 cookies which use at least one case of characters, digits and punctuation symbols. Overall, we conclude that the password strength is not an effective feature to identify authentication cookies. As to the entropy H and the index of coincidence IC, we observed that both of them work quite well in practice, though we do not present any plot of their distributions due to space constraints.

– *exp_date(c).* A cookie can be assigned an expiration date by the remote server, 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

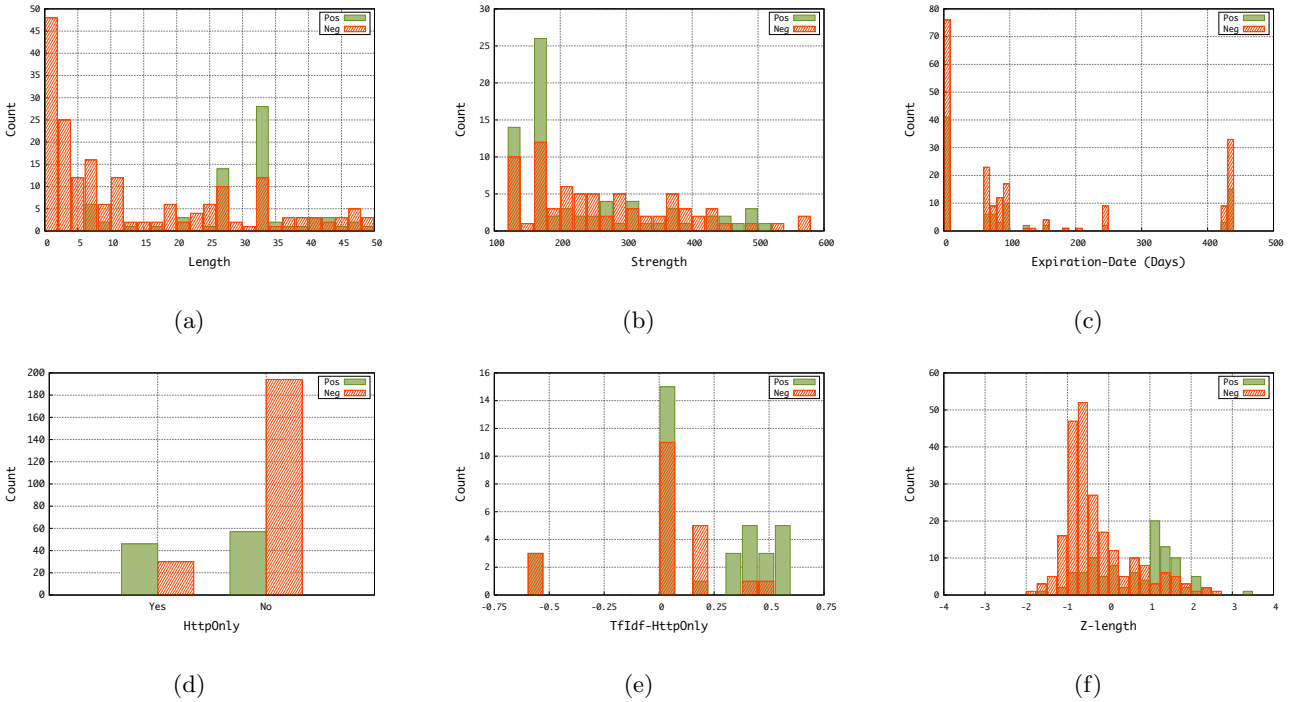

Figure 1: Feature distributions over cookie classes

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, as it is highlighted by the distribution plotted in Figure 1(c). It seems that different choices correspond to different web application semantics: security-critical websites (e.g., Dropbox) typically adopt authentication cookies with an early expiration date, while websites with more relaxed security requirements (e.g., Twitter) often release authentication cookies which expire much later, thus enhancing the experience of users who do need to authenticate with the website far less frequently.

5.2.2 *Novel Contextual Features*

After a careful evaluation of existing proposals, we manually inspected our gold set, 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 considering the characteristics of all the cookies $\mathcal{C}$ assigned 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.

– $tf\text{-}idf_{\texttt{HTTP-Only}}(c, \mathcal{C})$. Consider cookies 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 [6, 30, 18]. We show in Figure 1(d) the distribution of our two classes of cookies with respect to the presence of the `HTTP-Only` flag[3], 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 gold set 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, 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. Similar issues often arise in the area of information retrieval, where the effectiveness of a retrieval system depends also on accurately estimating the importance of words to each text document within a corpus. This idea is typically captured by the "term frequency-inverse document frequency" ($tf\text{-}idf$) score [25], 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

[3]The same analysis with similar results could be conducted on the `Secure` flag.

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 Figure 1(e) we show how HTTP-Only cookies distribute over the measure above. The plot highlights that several authentication cookies appear whenever $tf\text{-}idf_{\texttt{HTTP-Only}}$ is above a given positive threshold (~ 0.25).

– $Z_{length}(c, \mathcal{C})$. 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, as long as this allows for making all the authentication tokens invulnerable (cf. Definition 3.2). Hence, reasoning at the website level seems effective also to protect scenarios implementing the authentication scheme discussed before, and we thus argue to consider the *Z-score* [16] 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 mean and the standard deviation of the cookie lengths in $\mathcal{C}$, respectively. We show the distribution of the Z-score of the value length with respect to our two classes of cookies in Figure 1(f).

5.3 Choosing the Best Classifier

A key issue when designing supervised learning algorithms is that they should be able to find a tradeoff between *bias* and *variance* [11]. Roughly speaking, this refers to the ability of a learning algorithm to flexibly fit other datasets than the one on which it is trained (i.e., avoiding overfitting), while being able to perform effectively at least on the training set (i.e., preventing underfitting). It is now widely recognized that each algorithm has its own selective superiority, being best for some but not all tasks [5, 19].

In this work, we use the Weka toolkit[4], which is a collection of the most popular machine learning algorithms, to choose the best-performing binary classifier. The measures of validity used to assess the performance of each classifier are those we introduced in Section 4, namely *specificity* and *sensitivity*. Concretely, we use the gold set of cookies as described in Section 3 as our running dataset, and the set of features summarized in Table 6 to represent each cookie. Those include the most discriminating features discussed above as well as other two non-contextual features which have proven useful for classification.

To avoid overfitting, we adopt a procedure known as k-fold cross-validation [7], whereby the original dataset is randomly partitioned into k equally-sized subsets: $k-1$ subsets are used as training data to learn a classification model, and the remaining subset is retained as the validation data to test it. The cross-validation process is repeated k times (folds), with each of the k subsets used exactly once as the validation data: the k results from the folds are then averaged to produce a single performance estimation. Instead of a purely random partitioning of the dataset, we use a *stratified* 10-fold cross-validation strategy, which ensures the same class distributions in each fold as in the original dataset.

Eventually, we train several classifiers and the best performing turns out to be the one which builds a *functional tree*, henceforth referred to as FT. In a nutshell, this is a classification tree where each inner node and/or leaf may contain a test based on a combination (i.e., a function) of multiple features, as opposed to traditional decision trees [4, 20], whose nodes represent tests on individual features.

5.4 Experimental Results

We start by showing the confusion matrix obtained from FT and given in Table 7 below.

		Predicted	
		positive	*negative*
Actual	*positive*	*tp*: 83	*fn*: 20
	negative	*fp*: 33	*tn*: 191

Table 7: Confusion matrix for FT

We immediately notice that our machine learning solution performs significantly better than the examined hand-coded heuristics, obtaining a remarkable trade-off between sensitivity and specificity. Indeed, FT is able to limit the total number of false negatives to 20, which in fact is only worse than what is obtained by SessionShield, but is much better than all the other approaches (no less than 35 false negatives). The number of false positives evaluates to 33, which is better than *all* the other four examined solutions; in particular, FT outperforms the detector employed by SessionShield in this respect, since the latter produced 105 false positives in our dataset.

Overall, FT guarantees a high sensitivity (81%), which implies that it ensures a strong level of protection. At the same time, the specificity of our approach is very satisfactory (85%) and thus we argue that our classifier could be deployed inside a browser-based defensive solution without sacrificing the user experience. To summarize our results, we plot in Figure 2 the specificity and the sensitivity of all the examined solutions, as well as the *F-measure*, which provides a clear evidence of the improvement by combining the two scores above through their harmonic mean.

[4] http://www.cs.waikato.ac.nz/ml/weka/

Feature	Description	Type	Contextual
$length(v)$	number of characters in v	`numeric`	
$Z_{length}(c, \mathcal{C})$	the Z-score of $length(v)$ with respect to $\mathcal{C}$	`numeric`	✔
$IC(v)$	the index of coincidence of v	`numeric`	
$auth(k)$	k contains authentication-related terms (e.g., "*sess*")	`boolean`	
$http\text{-}only(c)$	c is flagged as `HTTP-Only`	`boolean`	
$tf\text{-}idf_{\texttt{HTTP-Only}}(c, \mathcal{C})$	the *tf-idf* of the `HTTP-Only` flag in c with respect to $\mathcal{C}$	`numeric`	✔
$secure(c)$	c is flagged as `Secure`	`boolean`	
$tf\text{-}idf_{\texttt{Secure}}(c, \mathcal{C})$	the *tf-idf* of the `Secure` flag in c with respect to $\mathcal{C}$	`numeric`	✔
$words(v)$	number of dictionary words in v	`numeric`	
$path(c)$	the path of c is '/'	`boolean`	
$timestamp(v)$	v contains a timestamp	`boolean`	

Table 6: Features used to classify the cookie $c = (k, v)$ within the context $\mathcal{C}$

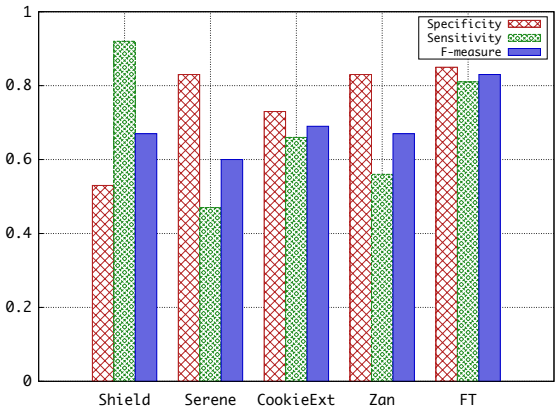

Figure 2: Performance evaluation

We conclude our experiments by performing an additional evaluation at the website level, much as we did for the detector employed by Zan. We observe that the 20 false negatives produced by FT are distributed inside 20 different authentication cookies from 19 websites. Overall, we identify only 12 vulnerable authentication tokens in 11 different websites: the improvement with respect to the detector employed by Zan is impressive, since the latter left room for attacks in 29 websites: the protection increases from 58.6% to 84.3% of the websites. Indeed, our classifier is very effective at protecting the authentication tokens of size 2 discussed in Section 5.2.1, where one cookie contains the username and the second one stores some password information. In particular, we observe that cookies containing password information are often correctly identified as authentication cookies by FT, which is enough to protect the authentication token.

6. RELATED WORK

In Section 4 we have already conducted an in-depth discussion of various existing client-side defenses for cookie-based sessions [6, 18, 24, 28]. Other browser-based protection mechanisms have also been proposed against different web security threats, most notably CSRF attacks [22, 23, 15]. 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.

A recent research paper sharing interesting similarities with our approach is [21]. 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 easily accounted for by simple checks on the browser-server interactions.

To the best of our knowledge, we are the first to leverage machine learning techniques to protect authentication cookies. Machine learning approaches, however, have been proposed in other areas of computer security, including intrusion detection systems [27] and spam filters [12].

7. CONCLUSION

We showed that existing authentication cookie detectors perform much worse than expected when they are tested against a gold set of authentication cookies, hence we concluded that any browser-based defense built on top of them is bound to provide an unsatisfactory level of protection. We advocated the adoption of machine learning techniques for the detection task and we showed a significant improvement with respect to state-of-the-art solutions. Compared to existing proposals, our solution provides a stronger degree of protection, while being precise enough to be actually deployed in practice.

Acknowledgements.

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. 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).

8. REFERENCES

[1] The Mechanize library for Python. `http://wwwsearch.sourceforge.net/mechanize/`.

[2] PHP session handling. `http://php.net/manual/en/book.session.php`.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *International Conference on Very Large Data Bases (VLDB)*, pages 487–499, 1994.

[4] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, NY, 1984.

[5] C. E. Brodley and P. E. Utgoff. Multivariate decision trees. *Machine Learning*, 19(1):45–77, 1995.

[6] M. Bugliesi, S. Calzavara, R. Focardi, and W. Khan. Automatic and robust client-side protection for cookie-based sessions. In *Engineering Secure Software and Systems (ESSoS)*, 2014. To appear.

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

[8] D. A. F. Florêncio and C. Herley. A large-scale study of web password habits. In *International Conference on World Wide Web (WWW)*, pages 657–666, 2007.

[9] S. Fogie, J. Grossman, R. Hansen, A. Rager, and P. D. Petkov. *XSS Attacks: Cross Site Scripting Exploits and Defense*. Syngress Publishing, 2007.

[10] W. F. Friedman. *The index of coincidence and its applications to cryptanalysis*. Cryptographic Series, 1922.

[11] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, January 1992.

[12] T. S. Guzella and W. M. Caminhas. A review of machine learning approaches to spam filtering. *Expert Systems with Applications*, 36(7):10206–10222, 2009.

[13] C. Jackson and A. Barth. ForceHTTPS: protecting high-security web sites from network attacks. In *International Conference on World Wide Web (WWW)*, pages 525–534, 2008.

[14] M. Johns, B. Braun, M. Schrank, and J. Posegga. Reliable protection against session fixation attacks. In *ACM Symposium on Applied Computing (SAC)*, pages 1531–1537, 2011.

[15] M. Johns and J. Winter. Requestrodeo: client side protection against session riding. *Proceedings of the OWASP Europe Conference*, pages 5–17, 2006.

[16] E. Kreyszig. *Advanced Engineering Mathematics*. Wiley, 4 edition, March 1979.

[17] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[18] N. Nikiforakis, W. Meert, Y. Younan, M. Johns, and W. Joosen. Sessionshield: Lightweight protection against session hijacking. In *Engineering Secure Software and Systems (ESSoS)*, pages 87–100, 2011.

[19] C. Perlich, F. Provost, and J. S. Simonoff. Tree induction vs. logistic regression: a learning-curve analysis. *Journal of Machine Learning Research*, 4:211–255, December 2003.

[20] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993.

[21] F. Roesner, T. Kohno, and D. Wetherall. Detecting and defending against third-party tracking on the web. In *USENIX Conference on Networked Systems Design and Implementation (NSDI)*, pages 1–14, 2012.

[22] P. D. Ryck, L. Desmet, T. Heyman, F. Piessens, and W. Joosen. Csfire: Transparent client-side mitigation of malicious cross-domain requests. In *Engineering Secure Software and Systems (ESSoS)*, pages 18–34, 2010.

[23] P. D. Ryck, L. Desmet, W. Joosen, and F. Piessens. Automatic and precise client-side protection against CSRF attacks. In *European Symposium on Research in Computer Security (ESORICS)*, pages 100–116, 2011.

[24] P. D. Ryck, N. Nikiforakis, L. Desmet, F. Piessens, and W. Joosen. Serene: Self-reliant client-side protection against session fixation. In *Distributed Applications and Interoperable Systems (DAIS)*, pages 59–72, 2012.

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

[26] C. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 1948.

[27] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *IEEE Symposium on Security and Privacy*, pages 305–316, 2010.

[28] S. Tang, N. Dautenhahn, and S. T. King. Fortifying web-based applications automatically. In *ACM Conference on Computer and Communications Security (CCS)*, pages 615–626, 2011.

[29] L. G. Valiant. A theory of the learnable. *Commununications of the ACM*, 27(11):1134–1142, November 1984.

[30] Y. Zhou and D. Evans. Why aren't HTTP-Only cookies more widely deployed. In *Web 2.0 Security and Privacy Workshop (W2SP'10)*, 2010.