Measuring Web Session Security at Scale★

Stefano Calzavara[a], Hugo Jonker[b,c], Benjamin Krumnow[b,d,*], Alvise Rabitti[a]

[a] *Università Ca' Foscari Venezia, Italy*
[b] *Open University of the Netherlands*
[c] *Radboud University Nijmegen*
[d] *TH Köln, Steinmüllerallee 1, 51643 Gummersbach, Germany*

Abstract

Session management is a particularly delicate component of web applications, which might suffer from a range of severe security issues, including impersonation attacks. Unfortunately, the scope and significance of prior work on web session security in the wild are limited by the complexity of the attack surface and the challenges of automating the login process on existing websites. In the present article, we fill this gap by proposing the first comprehensive, large-scale web session security measurement based on *post-login* data. Our analysis is comprehensive in that it deals with all key aspects of web sessions, i.e., the login process, the logout process and the authentication cookie handling. Our automated approach analysed an extensive set of session management practices of over 6,000 sites where login was successful and authentication cookies could be automatically detected, uncovering a widespread adoption of insecure practices in the wild.

Keywords: Session security, Shepherd, black-box testing, web measurements, automated login, authentication
MSC-code: 68M25

1. Introduction

Web application security is a complex matter, with multiple facets and moving parts. A particularly delicate component of most web applications is *session management*, where a user operating a client (browser) authenticates at a web application to request access to security-sensitive functionality, e.g., a payment interface of an e-commerce website. Web sessions are normally established upon successful verification of valid access credentials (login) and implemented on top of *authentication cookies*. Unfortunately, despite their apparent simplicity, web

★Authors listed in alphabetical order.

*Corresponding author
Email addresses: stefano.calzavara@unive.it (Stefano Calzavara), hugo.jonker@ou.nl (Hugo Jonker), benjamin.krumnow@th-koeln.de (Benjamin Krumnow), alvise.rabitti@unive.it (Alvise Rabitti)

Preprint submitted to Computers & Security

sessions can suffer from a wide range of severe security flaws [1]. Insecure implementation practices in web sessions may even lead to impersonation attacks, where the attacker uses the victim's password or cookies to authenticate as the victim and get unconstrained access to her account.

Web security studies aim to better understand causes for insecure practices; they unveil faulty implementations and highlight misunderstood concepts [2, 3]. However, web session security studies have been fairly limited so far. Analyzing web session security requires authenticated access to web applications, which is a difficult process to automate [4]. Thus, prior work on web session security reported on either (i) small-scale precise measurements involving a significant amount of manual effort [5, 6, 7], or (ii) large-scale measurements based on unauthenticated access to web applications, which miss valuable information, e.g., the login and logout processes [8]. The only notable exception is a recent paper, which analyzed post-login web session security at scale, but only focused on session hijacking enabled by cookie theft [9]. This means that prior web security studies are too small in terms of analyzed sites [5, 6, 7], too imprecise because carried out without performing authentication [8] or too narrow because they only cover a limited set of web session security threats [9]; we further discuss and compare against prior work in Section 8.

In the present article, we fill the gap in prior studies by presenting the first *comprehensive evaluation* of web session security that is based on *post-login data* collected through an automated *large-scale measurement*. Our analysis is comprehensive because it deals with all key aspects of web sessions, i.e., the login process, the logout process and the authentication cookie handling. Note that all these parts of the session management logic may be subject to vulnerabilities:

1. Web session security requires passwords to be protected against leakage over HTTP and to be reasonably hard to guess. If passwords are not appropriately protected against disclosure, impersonation attacks become trivial to perform.

2. Once a session is terminated by logging out, it should be invalidated at the server-side to ensure that authentication cookies are not valid beyond their intended expiration. Also, security-sensitive information stored at the client should be removed to minimize the risk of privacy leakage.

3. Insecure cookie configurations can fatally undermine web session security. For example, if authentication cookies are leaked in clear over HTTP, their theft may enable impersonation attempts (session hijacking).

We build our work on top of the Shepherd framework [4] for automated post-login studies, which we extend to mechanize the logout process and include new traffic collection facilities. Our analysis is designed to be non-intrusive and ethical: we leverage existing access credentials of popular sites from the public BugMeNot[1] database and we check compliance with security best practices

[1] http://bugmenot.com

without actively mounting attacks when we might violate existing terms of services. Despite these necessary limitations, our analysis is valuable because it identifies widespread adoption of insecure session management practices in the wild. We arrive at this conclusion by analyzing data collected after authenticating to 6,124 top sites from the Tranco list [10]. More concretely, our study shows that the risk of impersonation attacks on the analyzed sites is significant: for example, we identify 909 (15%) sites where impersonation might be enabled by an insecure implementation of the login process and 1,398 (23%) sites where impersonation might be enabled by the lack of confidentiality of authentication cookies. In addition, we identify a number of sites which implement the logout functionality insecurely: specifically, 469 (8%) sites do not terminate sessions at the server upon logout, while 230 (4%) sites do not remove security-sensitive information from the client after logout. All the vulnerabilities reported in the present article have been responsibly disclosed to the respective site operators.

Contributions. To sum up, we contribute as follows:

1. We use Shepherd [4] to create a data set of traffic and client-side storage related to all phases of session security: logging in, post-login, logging out. For this task, we extend Shepherd in two ways. First, we add support for automated logging out. Second, we enhance Shepherd to capture targeted parts of the HTTP traffic. This enables Shepherd to make use of its understanding of the login / logout processes during traffic collection and support further security analyses.

2. We review an extensive set of web session security threats, focusing on three different angles: login security, post-login security and logout security. For each threat, we identify automated testing techniques amenable for a large-scale security measurement in the wild.

3. We apply these testing techniques to data collected from 6,124 sites of the Tranco list [10] where Shepherd successfully logged in. We analyse the results to shed light on the current state of session security on those sites, detecting a widespread adoption of insecure practices.

2. Background

In this section we clarify how web sessions are implemented, we introduce our threat model and we review relevant background on web session security.

2.1. Web Sessions

A web session is established when a user operating a client (normally a web browser) provides valid access credentials to a web application by the submission of a login form, which is sent to a remote endpoint (the form's *action*) for verification. Normally, upon a successful verification of the access credentials, the web application issues a set of *cookies* which authenticate the user on the following HTTP requests [11], e.g., because they store a unique session identifier

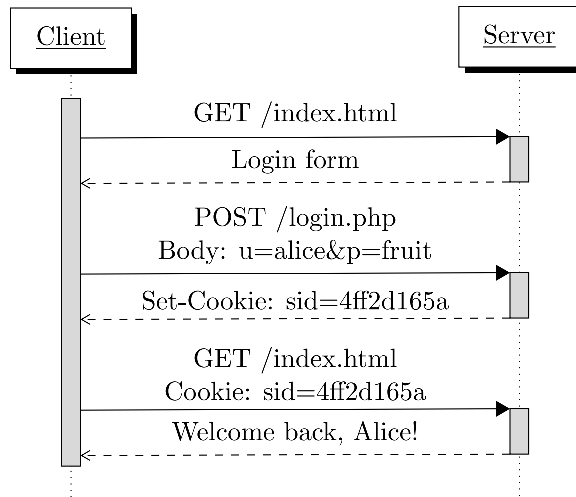

Figure 1: Example of web session

bound to the user's identity. Such cookies are known as *authentication cookies*[2] and are automatically sent by the client to the web application which set them.

Figure 1 shows the typical establishment of a web session, where the user Alice first logs in with password "fruit" and then remains authenticated by presenting an authentication cookie sid, which uniquely identifies her session (4ff2d165a). Once Alice has finished interacting with the web application, she can log out and move back to an unauthenticated state (not shown in the figure). This makes her session identifier invalid for future accesses.

2.2. Threat Model

We audit the security of web sessions against the traditional threats posed by *web attackers* and *network attackers*, the standard attacker models of the web security literature [12], which have been commonly used in previous web session security studies, e.g., [13, 14, 8, 15, 1, 16, 17, 9]. A web attacker is an unprivileged web user who operates a browser and has control of a malicious website. A network attacker extends the capabilities of a web attacker with the ability to inspect and arbitrarily modify the content of the HTTP traffic exchanged between the client and the server, e.g., because the attacker has control of the WiFi access point used by the client and operates from a man-in-the-middle position. However, a network attacker cannot sniff or corrupt the content of HTTPS traffic, assuming the adoption of robust cryptography and the deployment of a trusted certificate on the server. In our analysis, we only focus

[2] This is interchangeable with the term *session cookies* in some other work. We avoid the use of the latter term, since it can also be used to denote those cookies which are deleted when the browser is closed.

on sites equipped with certificates signed by a trusted certification authority according to a major commercial browser (Google Chrome). We also assume perfect cryptography, in the sense that our analysis focuses on session security, not cryptographic security of HTTPS. Note that cryptographic weaknesses in
115 HTTPS implementations are generally harder both to identify and to exploit in practice [16].

Finally, for the specific case of logout security, we also consider a *next user attacker*, who gains access to the client after the previous user has logged out of her session. This attacker covers often overlooked threats related to sharing
120 devices, such as borrowing someone's computer or using an Internet cafe. The next user attacker has access to the same browser and resources as used by the victim. More specifically, a website that does not clean up client-side storage upon logging out leaves behind information in the form of cookies and localStorage items. Information in sessionStorage is safe, because sessionStorage is
125 deleted when the user closes the corresponding browser tab.

2.3. Web Defenses

We review here a few common defenses designed to improve the security of web sessions.

2.3.1. Cookie Attributes and Prefixes

130 To understand the security implications of cookies, it is important to review their semantics. By default, cookies are only attached to requests sent to the same host which set them. However, a host may also set cookies for a parent domain by means of the `Domain` attribute, as long as the parent domain does not occur in the Public Suffix List:[3] these cookies, called *domain cookies*, are
135 shared across all the sub-domains of such domain. For instance, `a.foo.com` can set a cookie with the `Domain` attribute set to `foo.com`, which would also be sent to `b.foo.com`.

Cookies are normally shared across all protocols and ports. For instance, cookies set by a secure connection to `https://www.foo.com` are attached to
140 insecure requests to `http://www.foo.com`, i.e., they can potentially be stolen by network sniffing. To improve their confidentiality guarantees, cookies can be marked with the `Secure` attribute, which instructs browsers to communicate such cookies only over HTTPS connections. Similarly, cookies can be shielded from JavaScript accesses by marking them with the `HttpOnly` attribute, which
145 mitigates the dangers coming from script injection (XSS).

The lack of cookie isolation between protocols also implies that `http://www.example.com` can set cookies for `https://www.example.com`, i.e., cookies lack integrity against network attackers [14]. To avoid this, cookies can make use of the security prefixes `__Secure-` and `__Host-`. Though the semantics of the
150 two prefixes is different, both of them require the cookie to be set over HTTPS connections, thus providing cookie integrity.

[3] Available at `https://publicsuffix.org/`

2.3.2. HTTP Strict Transport Security (HSTS)

HSTS is a security policy implemented in all modern browsers, which allows hosts to require browsers to communicate with them only over HTTPS. Specifically, HTTP requests to HSTS hosts are automatically upgraded to HTTPS by the browser before they are sent. This way, site operators can assume that HTTP is banned and reduce the attack surface. Note that HSTS provides better protection than a standard HTTPS deployment (without HSTS), because HTTP communication is entirely forbidden, hence network attackers cannot impersonate the (non-existing) HTTP version of the target site.

HSTS can be activated over HTTPS using the appropriate header, which must specify a `max-age` attribute expressing the duration of protection. Moreover, the header can set the `includeSubDomains` option, which extends the scope of HSTS to all subdomains. Rather than activating HSTS via headers, hosts may request to be included in the HSTS preload list of major web browsers,[4] so that HSTS is activated on them by default. HSTS can be deactivated by setting the `max-age` attribute to a non-positive value.

3. Data Collection

We now provide details about our data collection. We start by discussing recently emerged approaches to automate the collection of post-login data. Then, we describe our data collection process, which tool we use and explain our modifications to it. Finally, we zoom in our data set and analyse its characteristics.

3.1. Access Credentials

The mandatory requirement for logging in across many websites is valid credentials. However, for legal and ethical reasons, leaked credentials cannot be used in our research. That leaves the following approaches to be considered:

1. using single sign-on (SSO)
2. automating registration
3. crowd-sourcing credentials

Note that none of these approaches will work flawlessly on all sites; each of these therefore introduces a bias in the set of sites covered by it. Some of this bias will be inherent to automated logins: credentials for, e.g., banking sites are not legitimately available at scale. Other bias will be specific to each approach.

Using SSO to log in is supported on 6.3% of the Alexa Top 1 Million [18]. SSO offers a clear advantage for large-scale studies, i.e., only a limited number of credentials are needed to log in on many different sites. Unfortunately, using SSO is also challenging: it may necessitate additional actions, such as account registration despite SSO access and authorization granting, e.g., in the case of

[4] Available at https://hstspreload.org/

OAuth 2.0. This makes using SSO rather hard. For example, Zhou and Evans had limited success [19]: 912 logins out of 20K sites (4.6%). In addition to those challenges, using SSO imposes its own bias on the set of sites: first, sites may insist upon their own account registration system and not offer any other login (e.g., webshops, banks). Other sites may not offer SSO for privacy reasons (e.g., adult entertainment). All such sites are excluded from an SSO-only approach. Moreover, there is no single, world-wide most popular SSO provider. Different regions prefer different SSO providers. Using common western SSO providers would bias the study towards their sphere of influence; minimising such bias necessitates a world-wide view on all SSO providers and their sphere of influence.

Automating account registration may address such concerns. A significant benefit of this approach is its general applicability, as it does not require SSO availability. In a recent study [9], this approach was used to login on 23,176 sites (out of 1.6M sites, 1.6%). A major downside to automatic registration is that the registration process is a critical security feature of websites frequently targeted for automated attack. As such, it is typically protected against automated visitors (e.g., by means of a CAPTCHA). Automating circumvention of techniques deliberately employed to prevent automated registration poses serious ethical concerns. Moreover, even if the ethical issues are ignored (we stress: they should not), automated registration still introduces a bias: it will only succeed on sites with insufficient defenses against it, thus likely skewing towards websites with weak security.

Using **crowd-sourced credentials** from public databases solves the ethical issues related to automated account registration. Nevertheless, this also leads to a bias. The bias inherent in legitimate crowd-sourced credentials is due to the type of accounts that users are willing or allowed to share. For example, sites where registration is simple and accounts are not associated with (personal) value will be prevalent, while other accounts (banks, social media, online stores), will be underrepresented or even absent due to the rules governing the crowd-sourcing effort. The current largest study based on this approach [4] gathered credentials for $\sim$50K sites, and was successful on 7.1K of these (14%).

To sum up, while automating registration managed (so far) to log in on the largest absolute number of sites, its success rate is an abysmal 1.6% [9]. Moreover, automated registration might violate existing terms of services, while still skewing the set of sites under consideration towards weak security. Using SSO is a more viable option, but requires a complex automation infrastructure to perform an open-ended scan with a low success rate (best success rate: 4.6%). In contrast, the use of crowd-sourced credentials minimizes the scanning effort and proved quite effective in the past (best success rate: $\sim$14%), which motivates its adoption in the present article. We acknowledge this approach might still suffer from a bias coming from the availability of credentials, which however is still not entirely solved by competitor approaches. In the article we thus report on several experiments designed to mitigate the impact of such bias.

3.2. Data Collection Tool

To collect data, we use Shepherd [4], a crawling framework based on Selenium and WebDriver to automate interaction with the Chromium browser. It uses a multi-step approach to automate the login process for unknown sites. First, it applies various strategies to identify login areas. Then, for each potential login area, it chooses a login routine based on the login areas characteristics. Next, Shepherd attempts to login with each given credential until it succeeds or all credentials were used unsuccessfully. If Shepherd believes a login succeeded, it verifies whether this is actually true. Finally, if successful login is verified, the same verification process is used to identify authentication cookies.

Input to Shepherd is a set of given URLs with site-specific credentials. On a set of unvetted credentials, Shepherd achieved a success rate of about 14%, which is the current state of the art, because the wide variety of websites makes a general automation of logging in challenging [4]. As discussed in the original paper, causes for failures are either due to invalid entries in the data set (e.g., sites without logins, invalid credentials), or part of the automated login process failing (login area not detected, CAPTCHA encountered, etc.).

3.3. Extending Shepherd

Shepherd [4] provides the login functionality needed for the present study. However, it does not support logging out or accessing network traffic, which are needed for our session security analysis. We extend Shepherd to include functionality for both. While capturing network traffic could be accomplished just by adding a proxy, a simple proxy would fail to account for Shepherd's awareness of where in the login / logout process it is.

3.3.1. Logout Automation

We leverage the similarities between the logout process and the login process, which is already supported by Shepherd. In particular, our Shepherd extension to log out follows similar steps, executed after a successful login.

The first step is to visit potential pages of interests. For our extension, we choose the page reached after logging in (likely a profile page) and the site landing page. Note that a well-designed website facilitates logout buttons on any page, after logging in. Second, we identify candidates for logout interaction elements. To this group belong elements that offer click functionality and contain keywords related to logging out. To determine if an element is clickable, Shepherd scans elements for attached event listeners, element tags (e.g., buttons, anchors etc.), and common properties of clickable elements. The third step is to define the order of elements to be triggered. For that, we rely on the distance of an element from the upper right-hand corner of the page. We noticed this aspect as a common property of logout elements during the development of our extension. This practice has also been shown to be successful for identifying login buttons in previous work [19]. Fifth, Shepherd triggers these elements first by opening URLs from anchor elements, and then by performing mouse clicks. The final action is the verification of successful logout. For a verification, Shepherd

visits the same page used to verify success of login, and checks whether login verification *fails*, i.e., the signals used to detect a failed login are used to detect a successful logout. More specifically, Shepherd uses the same information from the login phase to check whether the existence of login forms, logout elements, password fields and account information on the page has changed.

3.3.2. Capturing Network Traffic

The standard version of Shepherd provides access to a website's JavaScript, WebStorage items and cookies. However, it does not capture HTTP traffic, which is important for web session security analyses; for example, HTTP headers provide useful information about the adoption of defense mechanisms like HSTS.

Our goal is to analyse traffic related to specific phases of session management. Shepherd knows when each phase is reached and thus when traffic should be recorded. We therefore embed a way for Shepherd to enrich the recorded traffic stream with semantic information based on the selenium-wire package.[5] This enables our analysis to exactly target the various phases of session management, and opens the possibility to correlate website interactions (e.g., triggering a button, submitting a form and so on) with their corresponding network traffic.

In this project, we use this functionality in two ways. First, we let Shepherd mark the beginning and the end of each action of the traditional session management process (see Section 3.4.1). Second, we introduce marking for interaction steps, such as setting a marker when submitting a form and when the page has stabilized after form submission. This allows re-identification of traffic belonging to an action, which would be lost otherwise. We apply this functionality for traffic reduction. For that, we select actions (e.g., identifying the login page, false login attempts, etc.) that produce irrelevant traffic and remove them from our data set. We tested this in comparison to unfiltered traffic recording and found a reduction of captured traffic in size of up to 65%.

3.4. Data Collection Process

Like in the original Shepherd paper, we extracted the credentials used to access sites from BugMeNot[6], a website that provides crowd-sourced credentials for other sites. We searched BugMeNot for credentials for 1 million most popular websites according to the Tranco list [10][7], which aggregates the ranks from the lists provided by Alexa, Umbrella, Majestic and Quantcast from 14/4/2020 to 13/5/2020. The Tranco list is constructed to provide a more stable list of most popular websites, in contrast to individual rankings [10]. This resulted in a list of credentials for 56,437 websites.

3.4.1. Data Acquisition

We let Shepherd perform the following actions in sequence on these sites:

[5] https://pypi.org/project/selenium-wire/

[6] http://bugmenot.com/

[7] Available at https://tranco-list.eu/list/VKQN/1000000

action	# sites	out of	perc.
Connected	53,602	56,437	95%
Login area detected	35,465	53,602	66%
Failed login	29,699	35,465	81%
– *All credentials are invalid*	19,102	29,699	64%
– *CAPTCHA protects login*	2,676	29,699	9%
Logged in	6,766	13,687	49%
– *Authentication cookies identified*	6,124	6,766	91%
Logged out	3,302	6,124	54%

Table 1: Breakdown of the data collection process

connect → identify login area → log in → verify → visit subpages → derive authentication cookies → log out → perform security checks.

315 In addition to logging out and security tests, we included a step for deep scanning websites. Our goal is to capture authentication cookies that are not immediately set after logging in, or may only be set on subpages [6]. For that Shepherd extracts URLs from anchor elements that are embedded into the landing page. It first filters third party URLs and duplicates and then picks a random 320 selection of the remaining URLs. Shepherd limits its visits to a maximum of 5 subpages for performance reasons. We consider a subpage to belong to the same site when its URL shares the eTLD+1 of the site landing page.

Table 1 reports the number of sites reached for the different steps of the data acquisition process, as well as the number of failures for some automatically 325 detected failure cases with large impact. Shepherd's performance in our study roughly matches that discussed in its original paper, leading to a success rate of 13% [4]. Shepherd found a login area in 35,465 sites (66% of 53,602). Out of those, we found 19,102 sites where all the credentials from BugMeNot turned out to be invalid and 2,676 sites where the login process was protected by a 330 CAPTCHA, hence not amenable for automation. This leaves 13,687 sites where Shepherd had a chance to automate the login process, which succeeded in 6,766 (49%) cases. For most of these cases, we were able to successfully identify their authentication cookies as discussed below. In the following, we restrict our security analysis to the 6,124 sites where login was successful and Shepherd 335 could identify the authentication cookies.

During the data acquisition steps, we captured all requests and responses, with exception of the response body. This resulted in a data set of 86 GB. For each site, we captured cookies, LocalStorage and SessionStorage in four situations: (1) before logging in, (2) after verifying success of having logged in, (3) 340 after visiting several pages while logged in, and (4) after verifying success of having logged out. In addition, we keep track of which credentials were successfully used to log in, and what URL led to a login area. Once login is verified, we determine which cookies are authentication cookies, that is, cookies without which the browser is no longer logged in. Shepherd's initial implementation 345 relies on the work by Mundada et al. [6] and Calzavara et al. [20]. The worst

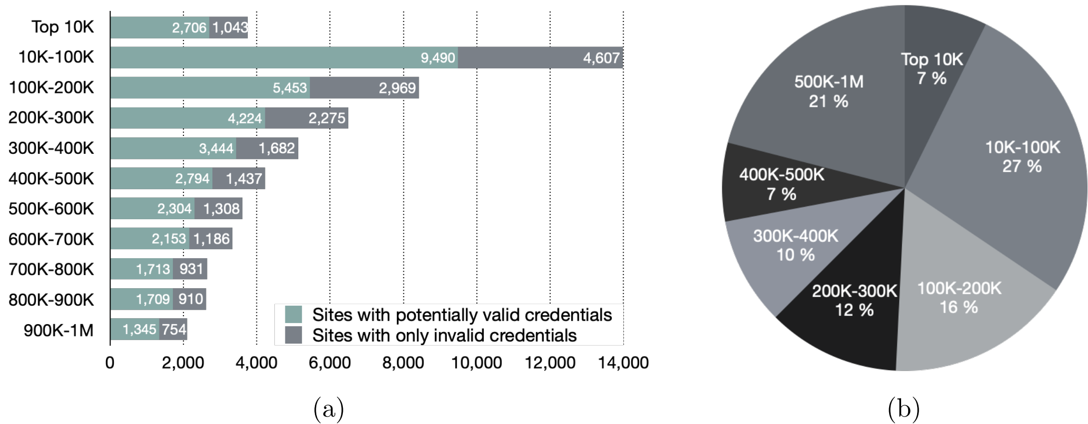

Figure 2: (a) Distribution of sites for which at least one set of credentials was acquired over the Tranco Top 1M; (b) Breakdown of successful logins by site popularity.

case scenario for this approach is an exponential run time with respect to the number of cookies. Therefore, we extended Shepherd to apply the improved solution by Calzavara et al. [21], which runs in linear time on most sites.

3.4.2. Significance and Potential Bias

350 With the respect to our discussion concerning the limitations of automated login approaches (cf. Sec 3.1), any research relying on such a data set should be checked for significance and biases.

To show that our data covers not just random sites from the tail of Tranco, but also very popular sites, we report two interesting results. First, Figure 2 355 (a) shows the distribution of sites for which at least one set of credentials was acquired over the Tranco Top 1M. The detection of invalid credentials is automatically done by Shepherd's "reasonably accurate" integrated detection routines [4]. The figure shows that the most popular sites from Tranco (Top 100K) are quite represented in BugMeNot. In contrast, Figure 2 (b) depicts the distri- 360 bution of sites with successful logins. This confirms that the data is distributed over the entire Top 1M, with more emphasis on the most popular sites and the first half of the Tranco list.

Next, we investigate the skewness of our data set. Due to the restriction on sites with a public login within our study, we expect an inherent bias. More 365 specifically: not all sites offer a login; such sites are inherently excluded from our study. Moreover, our credential source is crowd-sourced for the goal of avoiding login 'nags' – sites that pester visitors to create a login and limit content available to non-logged in users. We anticipate that this may cause certain types of sites to be underrepresented (e.g., malicious sites), and others, where login 370 nagging is common, to be overrepresented. To gain an estimate of this skewness, we derive categories for sites where Shepherd successfully logged in and compare it with categories of sites in the Tranco list. Specifically, we use Symantec's

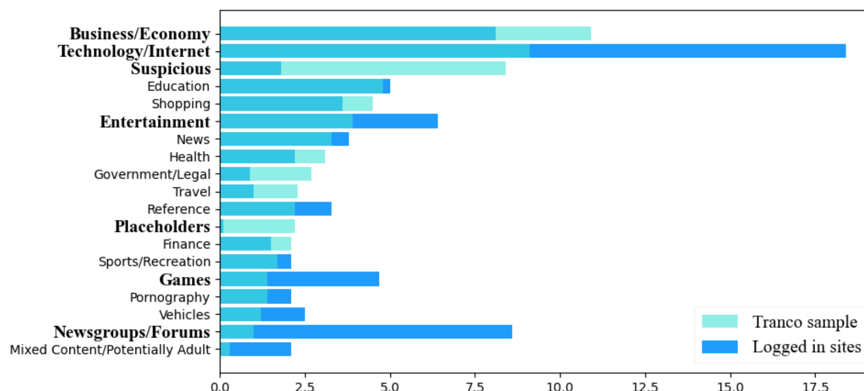

Figure 3: Relative frequency for all categories covering more than 2% of either our data set or the Tranco data set. Categories with a difference of over two percentage points between both sets are highlighted in bold.

Review Database [22] which classifies sites into 86 categories.[8] Unfortunately, access to Symantec's API is restricted through rate-limits, preventing us from 375 sampling the entire Tranco 1M list. We circumvent this restriction by creating a systematic sample of 50K sites (5% of the Tranco 1M list). We select domains based on a fixed interval (20 ranks), starting from a random position in the top 20 of the Tranco list.

Our results show that our Tranco sample contains sites from all 86 cate- 380 gories, while the login data set covers 79 categories. Notably, missing categories in our data set account for less than 0.1% of all sites. Figure 3 depicts the result for categories that exceed a 2% threshold for both data sets. Seven of these categories, marked in bold, differ by more than 2 percentage points between the sets. For these categories, we further discuss why these are over- or 385 underrepresented in our login set:

- **Sites requiring logging in by nature:** Some sites can only be used in their full potential when logging in. Unsurprisingly, we encounter such sites more frequently in our data set. Sites categorised as Games or Newsgroups/Forums are likely candidates that fit this description.
- 390 **Sites usually not shared by users:** Our goal is to investigate the security of legitimate sites targeted at genuine users. In our login data set, two types of sites occur rarely but make up for significant portion in the Tranco list: Suspicious and Placeholders. Since neither category brings value to users, these are less relevant to our study. Moreover, they

[8] https://sitereview.norton.com/#/category-descriptions

are also less relevant for genuine users and thus such sites are expected to occur only infrequently in a crowd-sourced data set. We find that this is indeed the case.

- **Tendency in the BugMeNot database:** Sites categorised as Technology/Internet and Entertainment are overrepresented in our login data set (by ∼9 percentage points). We believe this due to BugMeNot's mission and audience matching these types of sites particularly well.
- **Sites excluded by BugMeNot:** As prescribed in BugMeNot's terms of use[9], sites that offer paid content may not be submitted. This applies to certain sites in the Business/Economy category.

In conclusion, prevalence of categories in our data set mostly matches (within ±2%) incidence in the Tranco list. Deviations over this threshold are limited in number and small in size; we thus consider our data set to align sufficiently well with the Tranco set.

In more detail: only 4 out of 86 categories are significantly overrepresented. This is not surprising, as logins are not equally distributed over all categories. Finally, three categories are underrepresented: Business/Economy, Suspicious, and Placeholders. We consider the latter two less relevant for a security study, as neither are meant to provide genuine service to users. In particular, Placeholders sites do not concern real sites, but parked domains, search bait, etc. Similarly, Suspicious sites are sites that seem to be attacking genuine sites or users, not genuine sites themselves. This only leaves the Business/Economy category as underrepresented. The difference for this category is still relatively small (2.8 percentage points). Moreover, despite being underrepresented, it makes up for over 7.5% of our data set. Therefore, there is ample data for this particular category in our data set.

3.4.3. Failures in Logging Out

Careful readers would have noticed from Table 1 that automatically logging out from existing sites is surprisingly difficult: we only managed to automate the logout process on 3,302 sites, which is 54% of the sites where we successfully logged in. We manually investigated causes for failing logout. This revealed several causes. First of all, paths to logout elements vary more in labelling than for login elements. Some examples include logout, account, settings, profile, USERNAME, my SITENAME, etc. Exacerbating this, some websites hide the actual logout interaction element in overlay menus. That is, there are websites that only inject logout interaction elements into the DOM when the corresponding menu is activated. Identifying and triggering such menus is much more challenging, as these vary in appearance and implementation. Another cause we found is related to banned accounts. For these sites, logging in succeeds, but any interactive element in the post-login phase is blocked, including, interestingly

[9] `http://bugmenot.com/terms.php`

enough, the ability to log out. This problem is related to the used credentials, and cannot be resolved in the automation process. A third cause, seen in a a small number of sites, comes from confirmation requests when triggering a logout. Integrating handling of logout dialogues is left as future work.

4. Login Security

The security of web sessions can be broken when the password used for establishing the session is not appropriately protected. We consider two possible attack vectors, which would enable unconstrained impersonation of the victim: *password theft* and *password brute-forcing* enabled by insufficient password strength.

4.1. Password Theft

A number of insecure programming practices might lead to improper disclosure of passwords over HTTP. In particular, we focus on three prominent attack vectors:

1. If the action of the login form uses the HTTP protocol, the password is communicated in clear, hence even a passive network attacker who just sniffs the network traffic might disclose it. We identified 755 (12%) sites suffering from this vulnerability. Note that we implement this check on the actual login request available in our data set so as to minimize the number of false positives and false negatives, e.g., when the login form is submitted via JavaScript.

2. If the login page is served over HTTP, it can be modified by a network attacker so as to force password leakage, e.g., by changing the action of the login form to HTTP or by injecting an inline script which sends the password to the attacker's website. We identified 901 (15%) sites suffering from this vulnerability.

3. If the password is communicated in the query string of a GET request, it might become part of the URL of the landing page. This means that the password could be leaked as part of the `Referer` header if the landing page loads content over HTTP or from external sites. To spot such cases, we checked the `Referer` header of all the requests made during the website crawl, looking for our password value. We identified 4 sites leaking passwords to third parties (with Google servers being among the third parties in all cases) due to this vulnerability.

Overall, after removing overlaps between classes, we identified 909 (15%) sites exposed to the risk of password theft through the discussed attack vectors. Note that this number is dominated by the second case, i.e., login page served over HTTP. Notwithstanding the significant increase of HTTPS adoption in the last few years, insecurely served login pages remain a key factor of insecurity.

Two points here are worth mentioning about exploitation. First, modern browsers might implement security checks which prevent the introduction or communication of passwords in insecure contexts. However, such checks are not standardized and vary between different browsers, hence we consider bad practices like (1) and (2) as security issues. For example, we observed that while a recent version of Mozilla Firefox (80.0.1) warns users when they fill a login form which is going to be submitted over HTTP, this is not the case for a recent version of Google Chrome (85.0.4183). Moreover, a leakage of secrets via the `Referer` header might be prevented by appropriate configuration of the Referrer Policy header, which provides site operators with the ability of controlling the use of the `Referer` header.[10] However, due to our analysis methodology, we can confirm that all 4 vulnerable sites in the third class leak passwords to external sites via the `Referer` header.

Example: Chip PC

Chip PC Technologies (`www.chippc.com`) is a thin client manufacturer hosting a website to advertise and sell computers. The website provides access to a dashboard where customers can manage orders, warranties and licenses. While the website is served over HTTPS, the login form submits authentication credentials to `portal.chippc.com` over HTTP, hence even a passive network attacker can sniff passwords just by monitoring the HTTP traffic. This enables impersonation attempts, e.g., the attacker can access the victim's purchase history and steal her product licenses.

Example: World Wide Art Resource

World Wide Art Resource (`www.wwar.com`) is a website for artists and creatives who wish to publish their work, with optional paid tiers providing different content hosting plans, exposure and sales commissions. The website uses the GET method to communicate authentication credentials upon login, while importing several libraries from `google-analytics.com`, `consensu.org` and `sharethis.com` domains, in addition to some content from the affiliated website `www.absolutearts.com`. All these different hosts may get access to the passwords of logged in users through the `Referer` header of HTTP requests sent after login.

4.2. Password Brute-Forcing

Even if a password is securely transmitted from the client to the server, it can still be potentially disclosed by a determined attacker if it does not satisfy minimal password strength requirements. The French Data Protection Authority, CNIL, has issued recommendations for securing authentication. CNIL considers four cases, each with their own password requirements[11]: password only,

[10] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referrer-Policy

[11] https://www.cnil.fr/sites/default/files/atoms/files/recommandation_passwords_en.pdf

password + account access restrictions, password + additional authentication information, and two factor authentication. Of these cases, our approach can only succeed in logging in for the first two, hence we focus on them and observe
515 that, even in the presence of additional measures such as limiting the number of access attempts, CNIL recommends that the password must contain at least 8 characters from at least 3 of following sets: lowercase letters, uppercase letters, digits and special characters.

Unfortunately, there is no general automated way to detect which password
520 requirements are in place on a given site, since these are not necessarily explicit and can be enforced in different ways. To deal with this problem, we rely on two observations:

1. Although we cannot say anything about general password requirements, we can still check the password strength requirements on the password
525 used to access the web application under analysis, i.e., we can check whether our password is weak or not. This is valuable information for our measurement, since we did not create passwords ourselves, but rather used public passwords from the BugMeNot database, which can be used as a signal of inappropriate password requirements on existing sites.

530 2. HTML5 provides the `maxlength` attribute to enforce a maximal length for input elements, hence we can inspect its value to assess whether passwords are forced to be shorter than 8 characters. Moreover, HTML5 also supports the `pattern` attribute to enforce that inputs match a given regular expression, which can also be used to infer information about the general
535 shape of accepted passwords.

By combining these two observations, we identified 5,347 (87%) sites using passwords which do not satisfy minimal password strength requirements. The very large majority of our findings comes from the analysis of our own passwords, since the use of the `maxlength` and `pattern` attributes on password
540 fields does not provide much information. In particular, though we identified 884 sites making use of `maxlength` and 25 sites making use of `pattern`, we only found 3 sites where `maxlength` was used to limit a password field to less than 8 characters. The interesting point here is that we are guaranteed that, for those sites, all passwords are weak.

545 While the use of weak passwords is a bad security practice in general, it does not necessarily constitute an exploitable vulnerability. In particular, websites can implement detection or prevention techniques against brute-forcing attempts, such as locking accounts after a number of failed login attempts. We do not actively test for protection against brute-forcing at scale, as this is ethi-
550 cally dubious at best. In addition, it may violate a site's terms of services and put too much workload on the analyzed web applications.

Example: Geeks for Geeks

Geeks for Geeks (www.geeksforgeeks.org) is a popular portal offering articles on different technology-related topics, paid courses and hiring help. We

Site popularity	≤1K		≤10K		≤100K		≤1M	
Successful logins	53	100%	430	100%	2,081	100%	6,124	100%
Password theft	0	0%	12	3%	149	7%	909	15%
– login form sent over HTTP	0	0%	8	2%	103	5%	755	12%
– login page served over HTTP	0	0%	10	2%	146	7%	901	15%
– password in query string	0	0%	1	0%	2	0%	4	0%
Password brute-forcing	42	79%	363	84%	1,783	86%	5,347	87%

Table 2: Login security results by site popularity

555 have been able to access this site by using a BugMeNot password which is composed just by 4 lowercase letters. This implies that no meaningful password strength requirement is enforced on the website. This is concerning, because the odds of brute-forcing such passwords are realistically high, even if some kind of brute-force mitigation based on the frequency of failed attempts is put 560 in place.

4.3. Analysis by Popularity

Table 2 reports a breakdown of our analysis results by website popularity. The table shows two interesting observations. A positive result is that the most popular websites in our data set do not suffer from the risk of password 565 theft, since no site in the Top 1K leaks passwords in some way. However, the percentage of vulnerable sites monotonically increases when less popular sites are considered, up to a considerable amount (15%). This shows that the most popular sites have a more thorough HTTPS deployment than less popular sites, at least for the purpose of the login process.

570 Unfortunately, we also observe that the use of weak passwords is uniformly widespread and does not significantly correlate with site popularity: the number of vulnerable sites ranges from 79% to 87% in our popularity buckets. This might result from the bias coming from the use of public passwords from the BugMeNot database, since it is plausible that many security-critical sites with 575 strong password requirements are not included there. However, this does not undermine the significance of our finding: there are many popular sites which do not enforce minimal password strength requirements in the wild. Considered the massive user base of these sites, particularly in the Top 10k bucket, this result is both surprising and concerning.

580 5. Post-Login Security

Even when users rely on strong passwords which are appropriately protected, session security might be at harm due to the weak security guarantees of cookies in their default configuration. We first consider two traditional attack vectors: *session hijacking*, where the attacker impersonates the victim by stealing her 585 cookies, and *session fixation*, where the attacker impersonates the victim by forcing her to authenticate using a set of attacker-controlled cookies. Finally, we focus on two different types of *cookie brute-forcing* attacks.

5.1. Session Hijacking via Network Sniffing

Session hijacking happens when the attacker steals the authentication cookies of the victim and uses them to impersonate her at the target website. Recall that the current design of cookies leaves them susceptible to theft by network attackers, since cookies are normally shared between HTTP and HTTPS, hence potentially exposed in clear over the network. To avoid this, site operators can mark cookies with the `Secure` attribute, which restricts their scope to HTTPS. However, even cookies lacking the `Secure` attribute might be protected against disclosure over HTTP, in particular when the site uses HSTS to enforce the adoption of HTTPS at the client. We find a cookie to have *low confidentiality* against a network attacker when it lacks the `Secure` attribute and either of the following conditions holds true:

1. The server does not activate HSTS. In this case, the attacker can force an HTTP request to the site from the victim's browser and sniff the cookie in clear.

2. The cookie is set for a parent domain and the server activates HSTS without the `includeSubDomains` option. In this case, the attacker can force an HTTP request to a parent domain of the site to sniff the cookie in clear, as HSTS is only activated for the initial host.

	Host-only	Domain
Total	1,804	12,087
Lacks `Secure` flag	1,300	4,347
– low confidentiality	1,060	4,138

Table 3: Confidentiality properties of authentication cookies

Table 3 summarizes the confidentiality properties of the authentication cookies collected in our measurement. We observe that 59% and 34% of host-only and domain cookies respectively have low confidentiality against network attackers. Notably, most of the authentication cookies lacking the `Secure` attribute have low confidentiality, which suggests that the current state of the HSTS deployment in the wild is far from satisfying.

We say that a site is vulnerable to session hijacking when *all* its session cookies have low confidentiality, i.e., a network attacker can collect all information required to obtain the authentication cookies and impersonate the victim. In our data set, we identified 1,398 (23%) sites which are subject to this threat. Note that site operators might use defense-in-depth techniques, e.g., browser fingerprinting, to detect stolen session identifiers and terminate hijacked sessions. However, automating this analysis at scale would pose significant technical challenges: for example, sites might keep users authenticated and terminate sessions just when a security-sensitive operation is attempted. We acknowledge

this limitation and partially mitigate it by manually confirming successful session hijacking attempts on a random subset of 10 vulnerable sites, including the following.

625 *Example: Sotheby's*

The popular auction house Sotheby's runs a website (www.sothebys.com) that, while redirecting HTTP requests to HTTPS, does not serve any HSTS header, thus allowing requests to be sent over unencrypted connections. Since none of the site's authentication cookies is marked as `Secure`, a network attacker 630 can just sniff the first HTTP request sent to www.sothebys.com and gain access to valid session cookies. Note that the attacker could even force the browser to send such HTTP request by corrupting unrelated HTTP traffic received by the victim's browser.

5.2. Protecting Session Cookies from JavaScript Cookie Stealing

635 Web attackers may attempt session hijacking by stealing authentication cookies via JavaScript, e.g., exploiting an XSS vulnerability. To mitigate this threat, site operators should apply the `HttpOnly` attribute to their authentication cookies. For the same reasoning in the previous section, we consider a website as potentially vulnerable against session hijacking via JavaScript cookie 640 stealing when *all* its authentication cookies lack the `HttpOnly` attribute. We find out that out of 6,124 sites in our data set, 2,484 (41%) sites do not set this attribute for any authentication cookie.

Our analysis identifies sites whose authentication cookies lack inherent protection. Note that this lack of protection cannot be turned into an attack with- 645 out a script injection vulnerability. Nevertheless, it is relevant to analyse cookie protection itself, as XSS is consistently among the most common web security vulnerabilities [23]; furthermore, mitigation techniques like Content Security Policy fail to sufficiently address XSS in practice: up to 94% of policies in the wild do not protect against XSS [3].

650 *Example: Techrepublic*

Techrepublic (www.techrepublic.com) is an online news site within the Tranco Top 2K. It uses one cookie for authentication, which is protected against session hijacking attacks via the `Secure` cookie attribute and deployment of HSTS. However, the cookies are not protected against access via JavaScript. 655 This, by itself, does not quite enable session hijacking yet – only scripts in first-party context can access these cookies. Interestingly, Techrepublic includes several third parties in their first-party context, allowing these parties to access user authentication cookies. Finally, the lack of adequate protection of authentication cookies against JavaScript access means that protection against session 660 hijacking is fully dependent upon a flawless defense against XSS: any XSS flaw in the Techrepublic site can be leveraged to steal authentication cookies.

5.3. Session Fixation

Session fixation may happen when a website does not refresh the value of the authentication cookies when the privilege level of the session changes, e.g., upon login. In this case, the attacker can force a set of known authentication cookies from the target site into the victim's browser, so as to be able to impersonate the victim when the attacker later authenticates at the target and gets privileged access to it. To force cookies into the victim's browser, a network attacker can forge HTTP responses from the target site, thus abusing the lack of isolation between HTTP and HTTPS in cookie storage to eventually achieve the same effect as session hijacking.

While refreshing the value of authentication cookies upon login is a best practice, one can also thwart session fixation by ensuring the integrity of session cookies. Specifically, a cookie has *high integrity* against a network attacker when either of the following conditions holds true:

1. The server activates HSTS with the `includeSubDomains` option. In this case, the site forces the use of HTTPS on all the hosts which are allowed to set a cookie for it, thus closing the door to network attacks.

2. The cookie name contains a security prefix (`__Secure-` or `__Host-`), which can only be set and accessed over HTTPS.

We say that a site is vulnerable to session fixation when *none* of its session cookies is refreshed upon login and, in addition, *none* of them has high integrity. Interestingly, we found no authentication cookies making use of security prefixes in our data set. This outcome is in line with the observations of a recent study by Calzavara et al. [16], who found one site using cookie prefixes amongst 10K websites. We identified 1,082 (18%) sites which do not refresh authentication cookies upon login, including 1,011 (16%) sites which are deemed vulnerable to session fixation. The 71 sites which do not refresh authentication cookies, yet still are not vulnerable, all ensure cookie integrity by means of HSTS.

Example: Adult Entertainment Sites

We identified multiple adult entertainment sites vulnerable to session fixation attacks. In most cases, this comes from an inappropriate management of the PHP session cookie `PHPSESSID`. The default PHP session management does not account for logins, as the login logic is site specific. While PHP cannot refresh session identifiers upon login automatically, it offers the `session_regenerate_id` function to be invoked after login to prevent session fixation. It is concerning to find such vulnerabilities in adult entertainment sites, as a successful attack might leak sensitive information.

5.4. Cookie Brute-Forcing

We now focus our attention to two dangerous brute-forcing attacks on cookie values. The first threat we consider comes from the use of predictable identifiers in session cookies. The risk of brute-forcing attacks may be restricted by rate

limiting requests from the same client, or the expiration time of a session, in particular server-side session expiration. Unfortunately, the only way to test whether rate limiting is present, is to exceed the number of allowed requests. We refrain from such an unethical course. Testing server-side session expiration is also non-trivial. As shown in Section 6.1, client-side authentication cookies may officially expire long before the server-side session is removed. We are therefore left with considering to what extent cookie value itself is brute-forceable. We use the OWASP recommendations on session ID length,[12] which recommends session identifiers which contain at least 128 bits of entropy. We evaluate this by concatenating all authentication cookies, and computing the entropy of the resulting string. That is, we hold that, in these cases, the attacker can brute-force all the information required to get access to the victim's session. Our crawl identified 1,981 (32%) sites which do not satisfy this security best practice. The average value of entropy among the vulnerable sites is 92 bits, with a standard deviation of 39 bits.

The second threat we consider comes from an infamously insecure practice used for authentication cookie generation: computing the session identifier by applying a potentially invertible function to the password. This allows an attacker who gets access to a session identifier to recompute the password. This is a severe threat as it enables account takeover (via the password change interface) and might lead to impersonation on other services where the password is reused. In particular, we focus on two popular yet now insecure hashing algorithms: MD5 and SHA1. To identify these insecure practices, we compute the MD5 and SHA1 of the password we used to authenticate, and we look for them in the session cookie values. Overall, we identified 63 sites storing a weak hash of the password without salting inside a authentication cookie. Failure to use salting in hashing password results in far greater risk of offline/rainbow tables brute-forcing. We experimentally confirmed that 47 (75%) of these hashes can be trivially inverted into the correct password by using the CrackStation[13] rainbow tables free online service.

Example: DataLife Engine

We found 26 websites storing a weak MD5 hash of the password inside a cookie called `dle_password`, which is the authentication cookie of the DataLife Engine content management system. This is particularly concerning, because all sites built on top of DataLife Engine might improperly disclose passwords. In particular, we identified that in 15 cases the `dle_password` cookie could be sent in clear over HTTP: in 12 cases because the website was served over HTTP, in 3 cases due to the lack of the `Secure` attribute on an HTTPS website without HSTS. All these authentication cookies can be disclosed by network attackers and eventually inverted into the victim's password.

[12] https://owasp.org/www-community/vulnerabilities/Insufficient_Session-ID_Length

[13] https://crackstation.net/

Site popularity	≤1K		≤10K		≤100K		≤1M	
Successful logins	53	100%	430	100%	2,081	100%	6,124	100%
Session hijacking via network sniffing	9	17%	42	10%	358	17%	1,398	23%
Session hijacking via JavaScript	29	55%	192	45%	888	43%	2494	41%
Session fixation	6	11%	51	12%	312	15%	1,011	16%
Cookie brute-forcing	13	25%	100	23%	576	28%	2,044	33%
– weak session identifiers in cookies	13	25%	99	23%	564	27%	1,981	32%
– weak password hashes in cookies	0	0%	1	0%	12	1%	63	1%

Table 4: Cookie security results by site popularity

5.5. Analysis by Popularity

Table 4 reports a breakdown of our analysis results by website popularity. The key insight here is that there is no strong correlation between security and popularity. In particular, the percentage of vulnerable sites in the Top 1K bucket is only slightly lower than the percentage of vulnerable sites in the full data set, for all the vulnerabilities we considered.

We find this remarkable, because all the considered vulnerabilities are well-known and have easy solutions, hence we expected site operators at major companies to be aware of these problems and to be able to fix them. In retrospect, however, we see two possible reasons why top sites exhibit more positive figures for login security rather than for cookie security. First, understanding and enforcing login security is easier, since the adoption of HTTPS already fixes the most severe vulnerabilities. Considered how much HTTPS is getting traction, also thanks to the efforts by browser vendors, one might argue that login insecurity has been naturally fixed by the evolution of the web platform over the years. Moreover, based on our research experience, real-world web applications are complex and developed using a number of different technologies. This means that the session management logic is often spread through multiple authentication cookies issued by different components and it might be hard to assess the security of all of them.

6. Logout Security

Most websites offer users the possibility to terminate sessions by logging out. Though the logout process sounds simple in theory, there are a couple of implementation subtleties which might introduce security flaws. In particular, websites should properly implement both *server-side* and *client-side* session invalidation, as discussed in the following. Server-side session invalidation ensures that terminated sessions are forgotten by the server, i.e., presenting session cookies for those sessions should not enable authenticated access anymore. Client-side session invalidation, instead, guarantees that privacy-sensitive session information is removed from the browser upon session termination.

6.1. Server-Side Session Invalidation

The desired effect of a logout is that the session is no longer valid at the server side. If this is not handled properly, an attacker that manages to acquire session identifiers of incorrectly terminated sessions can still get authenticated access to the website. Moreover, unnecessarily extended session validity make session identifiers more vulnerable to the threat of brute-forcing.

In general, checking whether a website has proper server-side session hygiene consists of three steps: (1) login and keep cookies, (2) logout and (3) re-visit the site with the previously stored cookies.

The timing between logging out and revisiting is important. In a properly implemented session management system, server-side session cleanup should (at the latest) coincide with the notification to the client that the session has terminated. However, to account for sites sending a "session terminated" message in parallel with cleaning up session data in their backend servers, we check server-side session invalidation at three different times:

1. Immediately, that is: directly upon page stabilisation[14] after a logout request was sent by the browser. This is how an ideal session management implementation should work.
2. After 5 minutes. This time frame accounts for possible concurrency issues upon session termination, e.g., the logout request needs to be propagated to multiple replicated databases storing session information.
3. After 10 days. This time frame allows us to identify websites where sessions are not invalidated within any reasonable threshold and are definitely at risk.

In the second case, we let Shepherd evaluate every minute if a session is still active. This evaluations stops when the session turns out to be invalid or the five-minute mark is reached. For the final test, Shepherd re-uses the cookie jar from the original login and repeats the login verification step 10 days later.

Overall, we count 2,601 (82%) websites where session cookies were correctly invalidated directly after logout. In addition, we found 97 (3%) sites that did not invalidate authentication cookies immediately, yet did so within five minutes. This shows that some tolerance is useful in this kind of analysis. The remaining 604 (18%) sites did not invalidate authentication cookies upon logout within five minutes. Of these, 471 (14%) sites also failed the third test: 10 days later, the session was still valid at the server. This is worrisome, because, if a user's authentication cookies are *ever* captured by an attacker, the attacker might control the user's account.

[14] A page is considered as stable, when all HTTP responses are fully loaded and the DOM has not been updated for two seconds.

Example: Flattr

Flattr (www.flattr.com) is micro-payment service in the Tranco Top 10K. It enables users to make small (potentially recurring) donations to individuals as a form of patronage. We found that Flattr's authentication cookies were still valid 10 days after logging out. This is unexpected, given the nature of the site (micro-payments). Luckily, Flattr uses several protection measures that prevent cookie stealing, which mitigates the impact of this vulnerability.

Example: Suedkurier

Suedkurier (suedkurier.de) is a German regional newspaper with logins (free registration). Using the authentication cookies from the successful login resulted in a logged in state, 10 days after logging out from that session. Moreover, we found that Suedkurier's session identifiers have low entropy. The combination of low entropy and absent server-side invalidation significantly exacerbates the threat of cookie brute-forcing attacks.

6.2. Client-Side Session Invalidation

Session invalidation on the client-side serves to avoid data leakage. For example, network attackers can use the attacks from Section 5.1 to capture cookies left behind on the client even after session termination. This may leak privacy-sensitive information in case this is contained inside cookies, e.g., an email address. We also consider threats posed by *next user attackers* with access to the same client of the victim, as discussed in Section 2.2.

To evaluate proper session clean up, we search for Personally Identifiable Information (PII) in cookies and localStorage items that remain after logging out. In particular, we look for username, email and password in localStorage and in cookie values – both in plain text and hashed with MD5 or SHA1. Note that in our data set, username and passwords sometimes coincide. Thus we cannot always distinguish if the username or password was stored.

Our analysis identified 230 (7%) sites persisting PII in client-side storage after logout. A breakdown of the results according to the different types of client-side storage are shown in Table 5. Column $\mathbf{C}_{net}$ counts cookies which are not protected against network sniffing, hence can be accessed by both types of attackers we consider. Column $\mathbf{C}_{loc}$ also includes cookies which are locally accessible to the next user attacker alone, while column **L** reports on localStorage items. The table shows that, in 186 of the 199 cases (94%), PII is stored in cookies without protection against a network attacker. Similarly worrying, some sites store passwords in cookies, and do not remove these after a logout. We manually verified cases with passwords, and found that insecurity was typically obvious from the cookie name (e.g., `PASSWORD` or `passwd[207860]`). Finally, we also observe that when PII is stored, its value is rarely obscured by means of hashing.

We compare these numbers with PII stored during the login phase. We encountered 756 sites with PII in cookies, which were properly removed upon logout in 557 (74%) cases. We also checked use of localStorage: out of 199 sites

	$\mathbf{C}_{net}$	$\mathbf{C}_{loc}$	**L**
Username			
– *regular username*	105	109	14
– *email address*	13	14	16
Password	2	2	0
credential*	58	64	17
MD5 username			
– *regular username*	2	2	0
– *email*	2	2	3
MD5 password	0	0	0
MD5 credential*	6	7	0

C_{net}: Cookies accessible by network attacker
C_{loc}: Cookies accessible by next user attacker
L: localStorage
* cases where username = password

Table 5: PII left at client-side after logout.

storing PII in localStorage, 151 (76%) sites properly cleaned up localStorage upon logout.

855 *Example: Drop APK*

Our study revealed a file hoster within the Tranco Top 20K, Drop APK (dropapk.com), that keeps track of the username in a user's cookie jar. This cookie is not removed after logging out. For Drop APK, knowledge of the username suffices to list all public files of a user (https://dropapk.to/users/{username}).
860 A next user attacker can exploit this to identify a previous user's username on DropAPK and browse through the public files the user stored on the service.

6.3. Analysis by Popularity

Table 6 reports a breakdown of our analysis results by website popularity. Though the number of sites where we performed our evaluation is relatively
865 small, particularly in the Top 1K bucket, we do not observe any significant correlation between security and popularity. We identified sites incorrectly implementing server-side session termination in all popularity buckets, roughly with the same percentages. Similarly, errors in client-side session invalidation are also fairly constant with respect to popularity (ignoring the limited data for
870 $\leq$1K).

Interestingly, in all popularity buckets, the next user attacker is only slightly more powerful than the network attacker. This confirms that even top sites often overlook the adoption of cookie protection mechanisms, even for privacy-sensitive cookies. This is concerning, because we expected operators of top sites
875 to be more familiar with the semantics of cookies and their insecure default configuration.

	≤1K		≤10K		≤100K		≤1M	
Logged out	15	100%	169	100%	975	100%	3,302	100%
Server-side invalidation:	13	87%	137	81%	819	84%	2,833	86%
– immediately	11	73%	116	69%	734	75%	2,601	79%
– within 5 minutes	1	7%	7	4%	37	4%	97	3%
– 5 minutes – 10 days	1	8%	14	8%	48	6%	135	4%
– unknown, > 10 days	2	13%	32	19%	156	16%	469	14%
Client-side left PII behind in:	3	20%	14	8%	78	8%	230	7%
– localStorage	1	7%	6	4%	26	3%	48	2%
– Cookies_{loc}	2	13%	8	5%	60	6%	199	6%
– Cookies_{net}	2	13%	8	5%	56	6%	186	6%

Table 6: Session invalidation results by site popularity

7. Perspective

Our approach successfully logged in on 6,124 sites and logged out from 3,302 sites. What we found was quite concerning, at all levels of the session management logic. As to the login phase, we observed insecure connections for sending the login form (15%) or receiving it (12%), passwords leaked to third parties due to being submitted via GET instead of POST (4 sites), widespread (87%) allowance of weak passwords. After login, we identified authentication cookies vulnerable to session hijacking (23%) or accessible via Javascript (41%), session fixation vulnerabilities (16%), weak session identifiers (32%) and invertible password hashes stored in cookies (47 sites). Finally, after logout, we found sessions still not invalidated even after 10 days (8%), and failures to purge PII-containing session data from local session storage (8%).

Despite the bias coming from the analysis of sites for which valid access credentials can be found in a public database like BugMeNot, our results paint a troubling picture of the current state of the Web, because most of sites we analyzed are unquestionably popular services ranking in the Tranco Top 100k [10]. Although all the vulnerabilities we identified are relatively well known to web security experts, they are not necessarily easy to deal with and we recommend actions at many different layers to improve on the current state of affairs.

The first observation we make is that the login process is arguably the easiest part to secure of the session management logic. Security-savvy web users can largely mitigate the dangers coming from insecure login pages. In particular, users can leverage password managers to generate strong passwords even for sites which accept weak passwords, and they can install popular browser extensions like HTTPS Everywhere[15] to force the adoption of HTTPS even on sites which do not deploy HSTS. We observe that browser vendors can play a major role to improve login security and they are already taking actions in this direction. For example, the most recent versions of Google Chrome warn users when passwords

[15] https://www.eff.org/https-everywhere

905 are communicated in clear over HTTP and most modern browsers already ship an integrated password manager. We think and hope that by further pushing these actions it will be possible to rule out insecure logins from the Web within a reasonable time frame.

Unfortunately, despite their apparent simplicity, web session security issues 910 occurring after login are much harder to fix. There are several reasons for this. First, cookies are opaque to both web users and browser vendors, so detecting authentication cookies to analyze (and automatically improve) their security guarantees requires custom heuristics [21]. In particular, the most effective heuristics operate online (via testing) and are not straightforward to imple- 915 ment in commercial browsers without sacrificing performance or compatibility with existing web applications. In principle, one could try to experiment with safe defaults, e.g., automatically promote all cookies to Secure, however such forms of client-side protection can break existing websites [8]. In the end, we believe that secure session management crucially relies on the intervention of 920 site operators, i.e., browser vendors and web users are limited in their range of actions. Automated security scanners like our extension of Shepherd are thus an important tool to improve the current state of web session security.

8. Related Work

Web session security is a wide research area, whose key contributions were 925 summarized in a relatively recent survey [1]. Here, we discuss selected prior work which is most closely related to ours, and we describe trends based on previously conducted session security evaluations.

8.1. Comparison with closely related work

Only two previous studies assessed web session security after logging in with 930 an (semi-)automated approach: a first study by Mundada et al [6], and a second study by Drakonakis et al. [9]. Table 7 compares the aspects investigated by these studies and ours. The study by Mundada et al. [6] uses a manual login approach; users carry out the login process, while the security assessment is automated. Due to the manual login process, their corpus is much smaller 935 than either Drakonakis et al.'s work, or ours: only 149 sites have been analyzed. Drakonakis et al.'s study relies on account creation and logging in with SSO. This approach to automatically logging in has a low success rate. They compensate for the low success rate by attempting logins on the largest number of sites of all three studies, i.e., around 1.6M sites.

940 With respect to security analyses, there are several noteworthy differences between these studies. The overlap between the security assessment of Drakonakis et al. and our work concerns session hijacking via network sniffing and protection against JavaScript cookie stealing. Though there is some overlap between Mundada et al.'s work and our security analysis in terms of threats, there 945 are significant differences with our work. Their work primarily focuses upon

	[6]	[9]	this work
Logging in stats			
– login	manual	automated	automated
– # of sites approached	149	1.6M	53.6K
– # of successful logins	149	25.2K	6.1K
Login security			
– password theft	–	–	✓
– password brute-forcing	–	–	✓
Post-login security			
– session hijacking via network sniffing	✓	✓	✓
– session hijacking via JavaScript	✓	✓	✓
– session fixation	–	–	✓
– cookie brute-forcing	✓	–	✓
Logout security			
– server-sided session invalidation	✓	–	✓
– session data clean-up	✓	–	✓
Privacy			
– personal data leakage	–	✓	–

Table 7: Comparison of post-login studies investigating aspects of session security in detail.

automated detection of session cookies, rather than measuring web session security at scale (they only focus on 149 sites, due to limited login automation). As such, they do not evaluate login security and session fixation.

Other studies focused on specific web session security problems. For example, session hijacking has been studied against different threat models, including web attackers [24], network attackers [5] and both [8]. Session fixation also got some attention by the research community, particularly with the design of possible defense mechanisms [25, 26]. In more recent work, Calzavara et al. proposed black-box testing strategies to identify security flaws in web sessions, including session hijacking and session fixation [7]. However, the experimental analyses in all these papers are either small-scale (in the order of tens of sites) or based on data collected without logging in, which limits the analysis surface and requires one to come up with unreliable heuristics for authentication cookie detection [21].

The only research study on login security on the Web is due to Van Acker et al. [15]. They also discuss bad practices which enable exploitation by network attackers, e.g., login pages served over HTTP or sending the password in clear. However, their analysis methodology is different from ours, since they collect login forms by inspecting the HTML rather than by dynamically monitoring form submissions, which is generally more precise. For example, dynamic monitoring naturally covers the case of form submission via JavaScript, which was not handled in [15].

Compared to the security of login pages, more attention was given to the creation of passwords which are resilient to brute-forcing attacks [27, 28, 29, 30]. In our work, we base our analysis on standard recommendations from CNIL, which appear to be widespread based on anecdotal evidence. For example, the

year	#sites	logs in	Cookie attributes		HSTS	Invalidation	
			HttpOnly	Secure		Server	Client
2010, [31]	50	✓	48%	–	–	–	–
2011, [24]	419K	–	22%	–	–	–	–
2012, [32]	64	✓	–	48%	–	69%	–
2015, [2]	1.1M	–	–	–	1%	–	–
2015, [8][1]							
– 2014, from [20]	70	✓	63%	8%	–	–	–
– 2015 #1	1K	–	28%	5%	–	–	–
– 2015 #2	≤100	✓	38%	20%	–	–	–
2016, [5]	26	✓	62%	–	–	–	47%
2016, [6]	149	✓	68%	57%	[2]57%	50%	91%
2016, [33]	22K	–	–	–	11%	–	–
2019, [9]	25K	✓	77%	57%	[2]58%	–	–
2020, this work	6K	✓	59%	[1]59%	[1,2]63%	79%	93%

[1]: Numbers are reported in cookies and not sites

[2]: Numbers apply only to sites without protection of the `Secure` cookie attribute

Table 8: Trends in adoption of security measures (in % of sites).

popular LastPass[16] password manager generates passwords which follow the CNIL password strength requirements in its default configuration.

8.2. Trends in adoption of security measures

975 In the last decade, several studies have presented data on the current state of selected aspects of session security. Approach, measurements taken, and interpretation all vary significantly between these studies. Nevertheless, there is some overlap in the underlying security measures they sampled. This allows us to determine adoption trends in the last decade. Table 8 lists findings from 980 earlier studies. In general, we find that adoption rates for these simple server-side security measures are slowly increasing, though still far from ubiquitous. We discuss trends for specific measures below.

Adoption of the HttpOnly cookie attribute. Data on the adoption of the `HttpOnly` cookie attribute has been reported in [5, 6, 9, 31, 24]. Since the reported num-985 bers vary with each study's data set, these should be considered as a rough indicator for adoption. Reports before 2016 point to a low adoption rate between 22% and 63% at most. In comparison, later studies indicate an increase to a rate between 59% and 77% in a best-case scenario.

Adoption of the Secure cookie attribute. Multiple reports [8, 32, 6, 9] provide 990 data on the adoption of the Secure cookie attribute. As these reports differ how they report results (e.g. for partial or all cookies, for the entire site), they are not directly comparable. Nevertheless, the overall trend is clearly upwards, from a low of 5% in 2015 [8] to a vastly improved – but still disconcertingly low – 59% in

[16] https://www.lastpass.com

this work. Do note that we find susceptibility to session hijacking significantly lower at 23%. This is due to websites that deploy the `Secure` cookie attribute to some (but not all) authentication cookies and due to the deployment of other security measures, such as HSTS.

Adoption of HTTP Strict Transport Security. Previous reports on the adoption of HSTS show an overall small adoption: 1% of sites found by Kranch and Bonneau [2] in 2015 and 11% by Sivakorn et al. [33] in 2016. Thankfully, adoption rates have picked up in recent years, culminating in a 63% adoption rate to date. Nevertheless, all studies that investigated HSTS consistently find that lack of a `Secure` cookie attribute is only rarely mitigated by HSTS.

Server-side session invalidation. To the best of our knowledge, there are only two prior studies providing data on session invalidation, the study by Bursztein et al. [32] and the study by Mundada et al. [6]. Both studies have limited a sample size: 64 sites and 149 sites, respectively. As such, we cannot extrapolate from these studies, but we do note that a lack of server-side invalidation frequently occurred in either study. Our results suggests that the trend is improving, though we still find every fifth site failing to properly invalidate authentication cookies on the server-side following logout.

Session data clean up after logging out. Sivakorn et al. [5] conducted an in-depth study for privacy leakage on a small number of websites. As such, they evaluated if various privacy leaks even occur after logging out. They found that 47% of the assessed sites delete cookies holding PII. Mundada et al. [6] also performed tests for client-side cleanups. In contrast, they looked for authentication cookies that remain on the client-side after logging out. In their study, 91% of sites removed such cookies. Our study shows that this issue has further decreased.

9. Conclusions

We set out to investigate the current state of web session security in the wild, by performing a comprehensive session security analysis based on post-login data collected at a large scale. We used the Shepherd framework for post-login studies to automate logins, and extended it to handle logouts and capture traffic for further analysis. We acquired the needed credentials from a crowd-sourced repository (BugMeNot). We analysed security of the login process, security of the session (and its cookies), and security of the logout process. This includes an analysis of password strength of accepted passwords in practice, and the first (to the best of our knowledge) large-scale analysis of session invalidation.

As future work, we plan to further improve the automation of the logout process based on the data collected in the present study. We also want to further extend the scale of our analysis by integrating support for SSO, which would allows us to collect information from sites which are not included in BugMeNot. Finally, we would like to investigate how to extend our security analysis to other attackers, e.g., web attackers, without biasing the results of our results

1035 towards overly conservative assumptions, e.g., that all web applications might suffer from XSS or other script injections.

References

[1] S. Calzavara, R. Focardi, M. Squarcina, M. Tempesta, Surviving the web: A journey into web session security, ACM Comput. Surv. 50 (2017) 13:1–
1040 13:34. doi:10.1145/3038923.

[2] M. Kranch, J. Bonneau, HTTPS in mid-air: An empirical study of strict transport security and key pinning, in: Proc. 22nd Network and Distributed System Security Symposium (NDSS'15), The Internet Society, 2015. doi:10.14722/ndss.2015.23162.

1045 [3] L. Weichselbaum, M. Spagnuolo, S. Lekies, A. Janc, CSP is dead, long live CSP! on the insecurity of whitelists and the future of content security policy, in: Proc. 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS'16), ACM, 2016, pp. 1376–1387. doi:10.1145/2976749.2978363.

1050 [4] H. Jonker, S. Karsch, B. Krumnow, M. Sleegers, Shepherd: a generic approach to automating website login, in: Proc. 2nd Workshop on Measurements, Attacks and Defenses for the Web (MADWEB'20), IEEE Computer Society, 2020, pp. 1–10. doi:10.14722/madweb.2020.23008.

[5] S. Sivakorn, I. Polakis, A. D. Keromytis, The cracked cookie jar: HTTP
1055 cookie hijacking and the exposure of private information, in: Proc. 37th IEEE Symposium on Security and Privacy (SP'16), IEEE Computer Society, 2016, pp. 724–742. doi:10.1109/SP.2016.49.

[6] Y. Mundada, N. Feamster, B. Krishnamurthy, Half-Baked Cookies: Hardening Cookie-Based Authentication for the Modern Web, in: Proc.
1060 11th Asia Conference on Computer and Communications Security (ASIACCS'16), ACM, 2016, pp. 675–685. doi:10.1145/2897845.2897889.

[7] S. Calzavara, A. Rabitti, M. Bugliesi, Sub-session hijacking on the web: Root causes and prevention, Journal of Computer Security 27 (2019) 233–257. doi:10.3233/JCS-181149.

1065 [8] M. Bugliesi, S. Calzavara, R. Focardi, W. Khan, CookiExt: Patching the browser against session hijacking attacks, J. Comput. Secur. 23 (2015) 509–537. doi:10.3233/JCS-150529.

[9] K. Drakonakis, S. Ioannidis, J. Polakis, The cookie hunter: Automated black-box auditing for web authentication and authorization flaws, in:
1070 Proc. 27th ACM SIGSAC Conference on Computer and Communications Security (CCS'20), ACM, 2020, pp. 1953–1970. doi:10.1145/3372297.3417869.

[10] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, W. Joosen, Tranco: A research-oriented top sites ranking hardened against manipulation, in: Proc. 26th Annual Network and Distributed System Security Symposium (NDSS'19), The Internet Society, 2019. doi:10.14722/ndss.2019.23386.

[11] D. Kristol, L. Montulli, RFC 2965: HTTP state management mechanism, https://www.ietf.org/rfc/rfc2965.txt, 2000.

[12] D. Akhawe, A. Barth, P. E. Lam, J. C. Mitchell, D. Song, Towards a formal foundation of web security, in: Proc. 23rd IEEE Computer Security Foundations Symposium (CSF'10), IEEE Computer Society, 2010, pp. 290–304. doi:10.1109/CSF.2010.27.

[13] P. Chen, N. Nikiforakis, C. Huygens, L. Desmet, A dangerous mix: Large-scale analysis of mixed-content websites, in: ISC, volume 7807 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 354–363. doi:10.1007/978-3-319-27659-5_25.

[14] X. Zheng, J. Jiang, J. Liang, H.-X. Duan, S. Chen, T. Wan, N. Weaver, Cookies lack integrity: Real-world implications., in: Proc. 24th USENIX Security Symposium (USENIX Security'15), USENIX Association, 2015, pp. 707–721. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/zheng.

[15] S. van Acker, D. Hausknecht, A. Sabelfeld, Measuring login webpage security, in: Proc. 32nd ACM SIGAPP Symposium On Applied Computing (SAC'17), ACM, 2017, pp. 1753–1760. doi:10.1145/3019612.3019798.

[16] S. Calzavara, R. Focardi, M. Nemec, A. Rabitti, M. Squarcina, Postcards from the post-HTTP world: Amplification of HTTPS vulnerabilities in the web ecosystem, in: Proc. 40th IEEE Symposium on Security and Privacy (SP'19), IEEE Computer Society, 2019, pp. 281–298. doi:10.1109/SP.2019.00053.

[17] M. Steffens, C. Rossow, M. Johns, B. Stock, Don't trust the locals: Investigating the prevalence of persistent client-side cross-site scripting in the wild, in: NDSS, The Internet Society, 2019. doi:https://dx.doi.org/10.14722/ndss.2019.23009.

[18] M. Ghasemisharif, A. Ramesh, S. Checkoway, C. Kanich, J. Polakis, O single sign-off, where art thou? an empirical analysis of single sign-on account hijacking and session management on the web, in: Proc. 27th USENIX Security Symposium (USENIX Security'18), USENIX Association, 2018, pp. 1475–1492. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/ghasemisharif.

[19] Y. Zhou, D. Evans, SSOScan: Automated testing of web applications for single sign-on vulnerabilities, in: Proc. 23rd USENIX Security Symposium (USENIX Security'14), USENIX Association, 2014, pp.

1115
495–510. URL: https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-zhou.pdf.

[20] S. Calzavara, G. Tolomei, M. Bugliesi, S. Orlando, Quite a mess in my cookie jar! leveraging machine learning to protect web authentication, in: Proc. 23rd International Conference on World Wide Web (WWW'14), ACM, 2014, pp. 189–200. doi:10.1145/2566486.2568047.

1120 [21] S. Calzavara, G. Tolomei, A. Casini, M. Bugliesi, S. Orlando, A supervised learning approach to protect client authentication on the web, ACM Transactions on the Web (TWEB) 9 (2015) 15:1–15:30. doi:10.1145/2754933.

[22] Symantec, Webpulse site review request, https://sitereview.norton.com/#/, 2021.

1125 [23] OWASP, OWASP top ten – 2017: The ten most critical web application security risks, 2017. URL: https://owasp.org/www-project-top-ten/2017/.

[24] N. Nikiforakis, W. Meert, Y. Younan, M. Johns, W. Joosen, SessionShield: Lightweight protection against session hijacking, in: Proc. 3rd Symposium
1130 on Engineering Secure Software and Systems (ESSoS'11), volume 6542 of *LNCS*, Springer, 2011, pp. 87–100. doi:10.1007/978-3-642-19125-1_7.

[25] M. Johns, B. Braun, M. Schrank, J. Posegga, Reliable protection against session fixation attacks, in: Proc. 26th ACM Symposium on Applied Computing (SAC'16), ACM, 2011, pp. 1531–1537. doi:10.1145/1982185.
1135 1982511.

[26] P. de Ryck, N. Nikiforakis, L. Desmet, F. Piessens, W. Joosen, Serene: self-reliant client-side protection against session fixation, in: Proc. IFIP International Conference on Distributed Applications and Interoperable Systems, volume 7272 of *LNCS*, Springer, 2012, pp. 59–72. doi:10.1007/
1140 978-3-642-30823-9_5.

[27] S. Houshmand, S. Aggarwal, Building better passwords using probabilistic techniques, in: Proc. 28th Annual Computer Security Applications Conference (ACSAC'12), ACM, 2012, pp. 109–118. doi:10.1145/2420950.2420966.

1145 [28] R. Shay, L. Bauer, N. Christin, L. F. Cranor, A. Forget, S. Komanduri, M. L. Mazurek, W. Melicher, S. M. Segreti, B. Ur, A spoonful of sugar?: The impact of guidance and feedback on password-creation behavior, in: Proc. 33rd ACM Conference on Human Factors in Computing Systems (CHI'15), ACM, 2015, pp. 2903–2912. doi:10.1145/2702123.2702586.

1150 [29] R. Shay, S. Komanduri, A. L. Durity, P. S. Huh, M. L. Mazurek, S. M. Segreti, B. Ur, L. Bauer, N. Christin, L. F. Cranor, Designing password policies for strength and usability, ACM Trans. Inf. Syst. Secur. 18 (2016) 13:1–13:34. doi:10.1145/2891411.

[30] Y. Shin, S. S. Woo, What is in your password? analyzing memorable and secure passwords using a tensor decomposition, in: Proc. 28th The Web Conference (WWW'19), ACM, 2019, pp. 3230–3236. doi:10.1145/3308558.3313690.

[31] Y. Zhou, D. Evans, Why aren't http-only cookies more widely deployed, Proceedings of 4th Web 2.0 Security and Privacy Workshop 2 (2010). https://www.cs.virginia.edu/~evans/pubs/w2sp2010/.

[32] E. Bursztein, C. Soman, D. Boneh, J. C. Mitchell, Sessionjuggler: secure web login from an untrusted terminal using session hijacking, in: WWW, ACM, 2012, pp. 321–330. doi:https://dl.acm.org/doi/10.1145/2187836.2187880.

[33] S. Sivakorn, A. D. Keromytis, J. Polakis, That's the way the cookie crumbles: Evaluating HTTPS enforcing mechanisms, in: WPES@CCS, ACM, 2016, pp. 71–81. doi:https://doi.org/10.1145/2994620.2994638.