Shepherd: a Generic Approach to Automating Website Login[⋆]

Hugo Jonker
Open Universiteit Nederland
Radboud Universiteit
hugo.jonker@ou.nl

Stefan Karsch
TH Köln
stefan.karsch@th-koeln.de

Benjamin Krumnow
TH Köln
Open Universiteit Nederland
benjamin.krumnow@th-koeln.de

Marc Sleegers
Open Universiteit Nederland
marc.sleegers@gmail.com

***Abstract*—To gauge adoption of web security measures, large-scale testing of website security is needed. However, the diversity of modern websites makes a structured approach to testing a daunting task. This is especially a problem with respect to logging in: there are many subtle deviations in the flow of the login process between websites. Current efforts investigating login security typically are semi-automated, requiring manual intervention which does not scale well. Hence, comprehensive studies of post-login areas have not been possible yet.**

In this paper, we introduce Shepherd, a generic framework for logging in on websites. Given credentials, it provides a fully automated attempt at logging in. We discuss various design challenges related to automatically identifying login areas, validating correct logins, and detecting incorrect credentials. The tool collects data on successes and failures for each of these.

We evaluate Shepherd's capabilities to login on thousands of sites, using unreliable, legitimately crowd-sourced credentials for a random selection from the Alexa Top websites list. Notwithstanding parked domains, invalid credentials, etc., Shepherd was able to automatically log in on 7,113 sites from this set, an order of magnitude beyond previous efforts at automating login.

I. Introduction

Security of online services must be regularly tested. This is not only needed to improve security of specific services, but also to gauge the state of adoption of security measures. For websites, an interesting paradox presents itself: a major security aspect is the login process, with further security aspects of interest accessible only to logged-in users. However, the login process may vary from website to website. Thus logging in automatically across a wide variety of sites is a daunting challenge – one we address in this paper.

Websites offer users the option to login, typically for one of two reasons: to access protected resources (such as a personal mailbox), or to participate in the website's community under a specific identity. In either case, security of the authentication process is of fundamental importance. Websites should ensure that an unauthorised attacker cannot steal or overtake the login (session hijacking). However, websites are often vulnerable to simple session hijacking attacks.

[⋆] Authors listed in alphabetic order.

Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2020
23 February 2020, San Diego, CA, USA
ISBN 1-891562-63-0
https://dx.doi.org/10.14722/madweb.2020.23008
www.ndss-symposium.org

For example, in 2010, the Firesheep browser plugin for Firefox [But10] trivialised one class of (already known) session hijacking attacks. While this one attack can easily be prevented, authentication cookies can still be stolen or leaked in a number of other ways. Nowadays, several simple mitigation measures exist which can be used to prevent a whole range of simple attacks against authentication. These include cookie flags that restrict when a browser sends a cookie, HTTP headers that enforce secure communications for all subsequent visits, etc. Sites that lack these measures are vulnerable to simple session hijacking, while sites that do have them will offer a base line of security. Manually assessing the security of a specific site is straightforward. Indeed, due to manual verification, we know that the sites affected by Firesheep shored up their defences. However, applying this process to all websites does not scale and is thus typically not performed. Case in point: we do not even know how many other sites are still open to the Firesheep attack – or simple variations thereof.

A similar open question concerns the uptake of modern security measures. For example, we do not know how many sites lack basic security measures (proper cookie flags and HTTP headers) for logged in users. Other open questions concern adoption of security and privacy-enhancing measures beyond those affecting session security, for which logging in is a prerequisite. To study such questions requires two ingredients:

1. a set of valid credentials,
2. successfully submitting those credentials.

While several efforts have investigated specific security aspects on a handful of sites, to date, most studies that evaluated the security of the authentication process relied on a combination of automation and manual labour or (like Firesheep) tailored their measurement to specific websites. Typically, the manual aspect focused on actually logging in. This presents a barrier to scaling up these investigations and addressing the aforementioned questions. To the best of our knowledge, the largest manual study to date that successfully reaches post-login stages used manual logins to evaluate 149 sites. Initial attempts at automating the login process relied on single sign-on (SSO) credentials (such as Facebook login) and reported success on 912 websites.

Contributions. The goal of our work is to study the feasibility of large-scale post-login studies without tailoring the automation to a specific login flow. The only automated approach previously reported is tailored to the Facebook single

	lacks automated login			*has automated login*			
	vAHS17	**GRC18**	**MFK16**	**RB14**	**CTC15**	**ZE14**	**Shepherd**
Automation:							
- finding login area	✓	✓	–	–	✓	✓	**✔**
- submitting credentials	–	–	–	✓	✓	✓	**✔**
- verifying logins	–	–	✓	–	✓	✓	**✔**
# website languages supported	10	?	?	?	?	1	**19**
# sites scanned	100K	1M	149	203	215	20K	**49K**
# reached post-login areas	n/a	n/a	149	203	215	912†	**7.1K**

?	number of supported languages unknown.
†	computed, as [ZE14] does not explicitly state this number.
n/a	not applicable.

TABLE I: Comparison of manual and (semi-)automated login studies.

sign-on process. As such, it is unknown to what extent it is possible to automate the login process in general. To that end, we make the following contributions.

- We present Shepherd, a framework for automatically logging in on websites and executing a scan.
- We identify challenges in identifying login areas, on validating correct logins, detecting invalid credentials, and provide approaches to handling each of these.
- We perform a scan to illustrate Shepherd's potential. Using credentials gathered from a legitimate, crowd-sourced effort, we successfully login on 7,113 websites. The case study shows Shepherd is able to study a number of sites an order of magnitude beyond any previous study.

II. Related work

Various related work has studied web login systems and session security. Some studies proposed solutions to secure session cookies [dRND+12], [NMY+11], [BCFK14], [TDK11]. However, lacking the ability to automatically log in, none of these studies could test their solutions on authentication cookies. They all evaluated their solutions against cookies set prior to logging in. Tang et al. [TDK11] explicitly shy away from automated logging in, and even see it as infeasible. Where previous studies needed to log in, they typically relied on manual intervention. While this approach enables studies to log in, manual intervention does not scale well and is an obstacle to repeatability.

For example, Mundada et al. [MFK16] use manual logins to automatically analyse security of the login process of 149 sites with a browser extension. They found several security risks in well-known sites such as Yahoo. However, their approach is not repeatable without their volunteer corps.

Various steps towards more automated approaches to logging in have been made. Both the study by Van Acker et al. [vAHS17] and the work of Ghasemisharif et al. [GRC+18] needed to identify the login area. Van Acker et al. studied the security of the login area, while Ghasemisharif et al. counted the prevalence of single sign-on (SSO) providers. Both studies automated the identification of login areas, using similar methods. Van Acker et al. evaluated the Alexa Top 100K and found 32K login pages vulnerable to man-in-the-middle attacks. Ghasemisharif et al. evaluate the Alexa Top 1M and found 58K websites offering SSO login.

To the best of our knowledge, only three previous studies achieved some success in automatically logging in on websites. Calzavara et al. [CTBO14] used a crawler that submits credentials on websites by taking an URL and a pair of username and password. The crawler then searches for login pages and assess if the login was successful based on the presence of the username or absence of login forms. This gave them access to 70 websites and at that time largest dataset of authentication cookies. They extended this dataset to 215 websites in an extended version of their previous work [CTC+15]. The two other studies used Facebook as single sign-on (SSO) provider to login in on websites. Robinson and Bonneau [RB14] manually performed the step of finding login pages. For their study, they collected sites that offer Facebook connect and automatically logged in on them by using Facebook credentials. Their focus was on what permissions a visited site obtains to the user's Facebook profile. As such, they did not check whether the login was successful, nor did they evaluate aspects of the visited site. In contrast, Zhou and Evans [ZE14] designed an approach to automatically log into websites with Facebook and scan for SSO-related implementation flaws. Their scanner "SSOScan" automates the search for a Facebook login button, the submission of credentials, the eventual filling on registration forms and the evaluation if a login was successful on English-speaking sites. On the U.S. Top 20K websites they found 1,660 sites providing Facebook login, which they investigated. For the Top 10K, the authors report 80% success rate for their method.

Table I presents a comparison between case studies from related work and our case study with Shepherd. Thus, the given number reached post-login areas in the table is an extrapolation from the success rate and websites with SSO login areas found by the authors.

In conclusion, in related work we see a variety of approaches to studying login systems, and we find several venues to explore this. First of all, we could choose to focus on single sign-on (SSO) logins. The benefit is that the login processes for any specific SSO provider would be mostly uniform. The downsides are that automating logins for one SSO provider does not necessarily help for automating a second one, and that SSO logins are not that common (5.8% of the Alexa Top

1M, according to [GRC+18]). On the other hand, Van Acker et al. [vAHS17] found that about 51% of websites in the Alexa Top 100K offer the option to log in. A generic framework for logging in thus should primarily focus on site-specific logins, although SSO logins may provide an interesting addition (see Appendix A).

This does introduce the problem of acquiring credentials. This could be addressed by automatic account creation. However, the abilities needed for automating logging in (e.g., finding and identifying correct form, applying workflows to submit such forms, etc.) are part of the abilities needed for automating account creation. Finally, automatic account creation is a different subject with its own challenges, not to mention ethical considerations. Thus, Shepherd requires a supply of credentials. Such a supply may be manually constructed, e.g. from volunteers, or may be acquired from a legitimate source, such as BugMeNot[1].

III. Automating logging in

Logging in is basically a sequential process, consisting of a number of steps. To automatically log in on a website, Shepherd follows the following steps:

1. identify the login starting point,
2. submit credentials,
3. check response to login attempt,
4. verify whether login was successful.

In addition to these steps, Shepherd also detects and keeps track of certain errors. This is because Shepherd uses a generic approach, which is not tailored to any specific login process. As such, it may make mistakes (login field not found) or encounter errors from external sources (site unreachable, CAPTCHA, invalid credentials). In effect, the process acts like a funnel, with each step acting as an imperfect filter. To gauge the accuracy of the filters themselves, Shepherd includes routines to detect a variety of errors.

In the rest of this section, the steps and error detection are discussed in more detail.

A. *Identifying the login starting point*

First, the login starting point of the target website must be found. Zhou and Evans [ZE14] approached this by relying on click events on release to trigger SSO login dialogues. In contrast, domain-specific logins may also be found by visiting URLs. From previous studies [CTC+15], [vAHS17], [GRC+18] five search strategies emerge: scanning the landing page, visiting URLs filter by login keywords, querying search engines, and scanning clickable DOM elements. We found that using multiple search engines can lead to a better coverage, thus we expanded Van Acker et al.'s approach on this point.

Unfortunately, none of these studies provide insight in the efficiency nor reasoning about the order of these methods. To test the success of each method for finding login elements, we applied each to a random sample of 5,000 sites from the Alexa Top 1M (cf. Table II). We found that landing pages rarely contain login elements. However, all other methods rely on the actual domain of the site (after any redirects). Therefore, the method of scanning the landing page should be executed first. Furthermore, some methods are more successful in finding login pages than others. Interestingly, the results of the various methods are sufficiently disjoint that combining them leads to highest success rate.

	Method	Success (n=5000)	Avg. Time (in sec)
1	Landing page	225	7.17
2	URLs with login terms (depth 1)	1,119	10.86
3	Clickable Elements	1,149	18.84
4	standard URLs	1,366	18.79
5	Search engines	1,948	50.08
	- Startpage.com	1,378	32.66
	- Bing.com	342	9.17
	- Ask.com	1,216	18.04
	All combined	2759	36.43

TABLE II: Performance overview of methods to locate a login page of a website

Based on the evaluation, we arrived at the following order to search for login elements:

1. Landing page,
2. URLs with login-based terms[2] found on the landing page,
3. Clickable elements with login-based terms,
4. Standard URLs[3],
5. Search engines,
6. URLS with login-based terms found on pages from step 2.

The order of these methods is important. Shepherd looks for login elements on the landing page first, since that page needed to be loaded anyway. Only when method 2 and 3 fail, Shepherd uses more generic methods. We gave standard URLs a lower priority, as these can lead to admin login pages. Shepherd only uses search engines if prior methods fail to reduce the risk of blocking and reliance on external parties. Finally, if none of these methods work, Shepherd scans each of the pages found in step 2 for URLs with login-related terms and visits these.

Once a method claims success, Shepherd stops searching for the login. If none of the search method worked, Shepherd finishes the scanning process and marks that a login page could not be found.

When Shepherd encounters a visible input element of type password which is not part of a registration form, it assigns the status *login found*. A form is considered a registration form if it contains more than 3 visible input elements (including the found password field).

B. *Submitting credentials to login*

Once the login element has been found, the credentials must be submitted. There are two common types of logins:

[1]A legitimate, crowd-sourced service to circumvent free-but-mandatory registration. Website inclusion is subject to certain legal restrictions (no age-restricted sites, no banks, etc.). Moreover, it offers site owners a simple interface for removing their site from the data set.

[2]Shepherd contains a dictionary with multiple translations for keywords from native speakers and Google translations.

[3]Specifically: http(s)://base_url/login, http(s)://base_url/account and http(s)://base_url/signin.

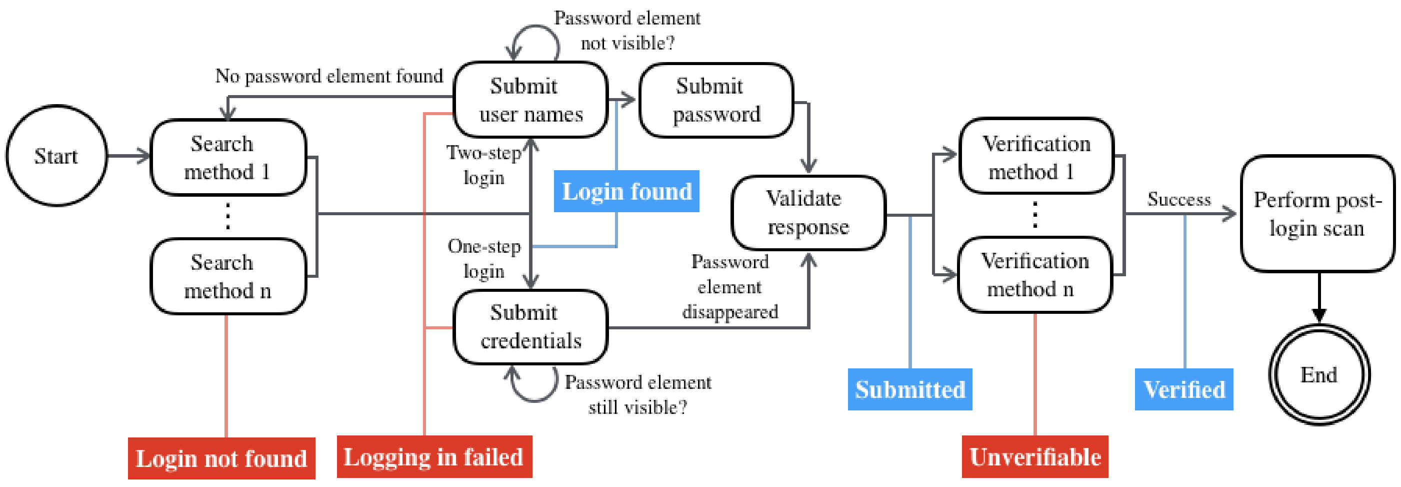

Fig. 1: Steps of the login process after connecting to a target site.

- one-step: where username and password can be supplied simultaneously; and
- two-step: first request username, and only after the username was submitted is the password entry field shown.

By combining logging in with scanning for login areas, we are able to cover both types. To the best of our knowledge, submitting credentials for two-step login has not been explored in previous work.

If fully submitting the credentials (in either one-step or two-step fashion) causes the password input element to disappear, the website status is set to *submitted*. This status indicates that the site responded to the input, but does not claim that login was successful. Websites can also stop showing password fields in other cases, such as when the user is blocked or an error results in a 404 page. To separate such cases from actual logins, the submission process is followed by a verification process.

Shepherd can use multiple credentials per domain (there is also limited support for Facebook SSO credentials, cf. Appendix A). If it is not successful with the first set of credentials (i.e., the password field remains), Shepherd will try logging in anew with the next set of credentials. In case all available credentials for a specific domain fail, the website is assigned the status *logging in failed, after trying all credentials*.

Finally, in some login forms, the username is an email address. If the input element is of type "email", Shepherd avoids submitting strings that are not valid email addresses. For cases, where Shepherd can ensure that an email address is required to login, given credentials without an email address will be ignored.

C. Checking the response to a login attempt

Shepherd checks the website response after submitting credentials in order to perform some error detection. For example, many websites signal a failed login attempt with a message like "username or password invalid." Shepherd detects such messages and marks the credentials as invalid. More specifically, Shepherd assesses a website's language and searches for visible strings containing sets of keywords, such as "invalid" and "username" or "password". Only if these terms appear combined in a single string, Shepherd marks the credentials as invalid. In practical tests, failure messages occasionally appeared in English on non-English sites, so Shepherd always scans for English keywords concerning invalid credentials. Besides invalid credentials, Shepherd also detects CAPTCHAs and blocking messages to recognise countermeasures against automated visitors. To do so, Shepherd scans visible elements for blocking or CAPTCHA related keywords in multiple languages, and checks the HTML source for code fragments pertaining to CAPTCHAs. The code fragments Shepherd detects are derived from the HTML code fragments used to invoke one of five CAPTCHA libraries (including ReCaptcha).

D. Verifying login status

After detecting that the website reacted to the submission of credentials, Shepherd evaluates whether logging in was successful. For this, we use previous approaches [CTC+15], [ZE14], [MFK16] with a few differences. In previous studies, trusted credentials or manual logins were used. In this work, we assume credentials to be unreliable. Moreover, we also do not assume that the occurrence of a string that matches the username on a page is sufficient to verify login – strings occurring in usernames may also occur on the page due to other reasons.

To verify a login, Shepherd runs a verification method twice: once on the potentially logged-in site, and once without cookies. Login is only successfully verified if the first check succeeds and the second fails.

Shepherd has three different verification methods used for this:

1. detect a logout button or user identifier on the page received following submission of credentials,
2. detect a logout button or user identifier on the landing page,
3. attempt to re-open the login area[4].

[4]In well-designed sites, this should not be possible for logged-in users.

A login is only claimed to be verified if at least one of these verification methods is successful when visiting the site with cookies, and fails when visiting the site without cookies.

E. Perform post-login scan

Finally, following successful verification of logged-in status, Shepherd will execute any scans (see Section IV-D).

IV. Implementation

Shepherd uses Chrome as a designated browser, and runs on Linux and on MacOS systems. A configuration setting determines whether a headful or headless browser is used for scanning. The instrumentation is achieved through usage of Selenium.

A. Base HTTP platform

There are several possible platforms on which to build Shepherd. Not all are suitable. Commandline tools and HTTP libraries lack engines to interpret JavaScript and construct DOMs, which is necessary for web sites with dynamic content. More advanced tools, such as high-level browser automation libraries (e.g., NightmareJS) or headless browsers (e.g., PhantomJS), are an improvement on this but nevertheless still lack some of the functionality of full browsers (e.g., plugins). This poses two problems: firstly, they do not necessarily provide a faithful rendition of what a regular user would experience (cf. [EN16]); secondly, such deviations will (also) affect logging in (cf. [JKV19]). Thus, for this project we require an automated way to use a regular (full) browser. Shepherd is based on the standard tool for this, Selenium. Selenium allows programmatic access to a variety of browsers.

B. Optimisation

Logins are typically slow and can easily take several seconds. When attempting to login on unknown sites, using a form which may or may not be the login form, with credentials that may or may not be valid, several passes have to be taken. When executing a study over many sites with all these factors in mind, performance becomes an important factor.

With respect to optimisation, we found that Selenium's built-in functions are slow compared to executing the same functions in JavaScript. For example, we measured that accessing the plain HTML content of elements takes 14 msec using Selenium functions. When operating with a large number of elements, this becomes a time-consuming operation. Another example is Selenium's function to query multiple elements *find_elements_by_css_selector()*, which takes 1 sec per query. Combining that with additional filtering based on an element's attributes or content results in a large overhead. Therefore, we switched from using Selenium's functions to using in-browser JavaScript.

To this end, Shepherd provides two JavaScript functions, *href_scanner()* and *element_scanner()*. These functions allow efficient selection of anchor and other elements. The former function searches amongst anchor elements with HREF attributes, while the latter can select any element through a custom CSS3 selector. Both scripts take a regular expression to filter selected elements based on their HTML content (e.g., login-related keywords).

Using these in-browser functions instead of Selenium functions provided a noticeable speedup. For one site, switching to JavaScript functions improved time for accessing and filtering elements from 16.8 sec to 50 msec.

C. Performance

With the above measures in place, Shepherd needs about 75 sec to scan a site. Thus, Shepherd can scan and login to about 1,500 sites per browser instance per day. In our experiments, we found that a regular end-user machine can run 5 browser instances, so Shepherd can scan about 7,500 sites per day per computer.

D. Post-login scanning

Following login, payload scans (implemented as Python modules) are executed. Shepherd provides an interface to interact with the browser and detect effects of interactions. This interface is a wrapper of Selenium commands, but streamlines error handling and ensures performance-optimised commands are used by the scanning module. In addition, Shepherd offers functionality to determine which cookies are authentication cookies based on algorithms used in earlier work [MFK16], [CTBO14]. Furthermore, additional modules can be hooked into Shepherd. This allows for sequential execution of several scanning modules. Scan results are determined on the fly and stored in CSV files for a posteriori analysis.

V. Evaluation: Logging in on websites in the wild

In this section, we evaluate Shepherd's ability to log in by means of a large-scale experiment.

A. Acquiring credentials

We created a specific crawler to extract credentials from BugMeNot. The crawler uses a list of domains and for each domain, extracts the credentials. Moreover, for each set of credentials, it also stores meta-information supplied by BugMeNot (success rate and number of votes). We seeded our crawler with the Alexa Top 1 Million sites of October 2018. This resulted in the extraction of 129,252 accounts for 49,846 unique domains.

The collected dataset covers over 37% of the Alexa Top 10K domains and around 18% of the Top 100K, respectively (see Figure 2). The concentration of websites decreases with the rankings and appears to converge around 2K domains per 100K websites.

B. Shepherd's login performance

Using dataset with credentials from BugMeNot, we can measure success rate and error causes. Starting with credentials for 49,846 sites, Shepherd was able to automatically detect that all available credentials for 23,088 sites were rejected by the site as invalid. This leaves 26,758 sites to attempt login. Shepherd could verify successful login on 7,113 sites, i.e., 26.6%. This is a lower bound: there will be external sources of errors that Shepherd failed to detect. For example, it is

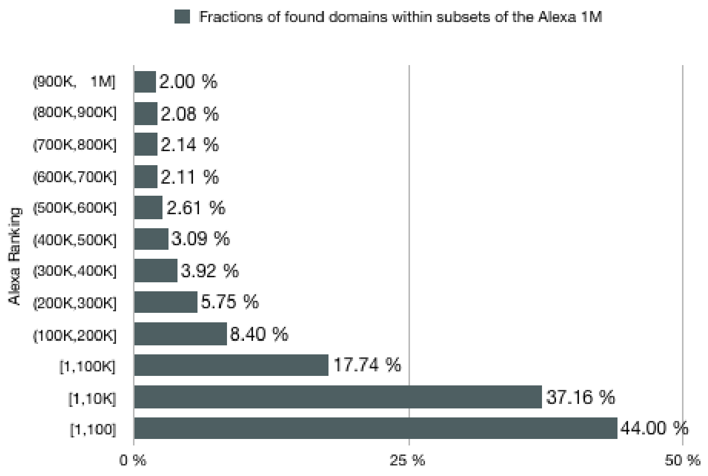

Fig. 2: Relative frequency of domains with credentials of our testing database within the Alexa Top 1 Million.

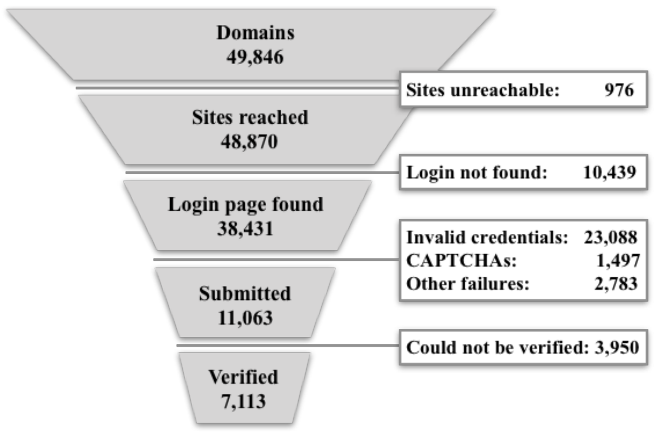

Fig. 3: Funnel of the login process for domain-specific credentials.

	# sites	out of	
Total	49,846	–	
Sites not reached	**976**	49,846	(**2.0%**)
Login page not found	10,439	48,870	(21.4%)
Login failures:			
- **Invalid credentials detected**	**23,088**	38,431	(**60.1%**)
- **CAPTCHAS**	**1,497**	38,431	(**3.9%**)
- Unaccounted failure	2,783	38,431	(7.2%)

TABLE III: Failures detected by Shepherd. Failures caused by external factors are marked in bold.

not certain that all websites in the set offered the option to login. Moreover, in some of the 3,950 cases where Shepherd managed to submit credentials, it may have been successful but failed to verify this.

While this leaves ample room for improvement, this case study is of unprecedented scale – easily an order of magnitude beyond any previous studies.

As discussed before, logging in is a sequential process, which means that an automated approach must execute sequentially. Imperfections in each step result in a funnel-alike propagation through the login sequence, depicted in Figure 3. The rates shown in Figure 3 were automatically detected (see also Table III). Of course, not all failures can be automatically attributed; for example, failing to reach the step *submitted* can be due to CAPTCHAs or invalid credentials, both of which are automatically detected by Shepherd. These error sources account for the bulk of the failures for moving from *login page found* to *submitted*. Nevertheless, there are 2,783 websites where this transition failed, yet the built-in failure attribution did not detect a cause.

Some of the failures are due to external causes, while other causes denote potential areas for improvement of Shepherd. Failures on 25,561 sites (51.3%) were attributable to external factors: site unreachable, no valid credentials, or CAPTCHAs. The major area for improvement is identifying the login starting point, which failed on 10,439 sites (20.9%).

VI. Validation of Shepherd

Note that due to the untrustworthy source of credentials, it is not certain that a website for which we possess credentials actually has a login facility. Also, our underlying heuristics are not 100% perfect and may occasionally fail to determine the status correctly. To determine bounds on the error rates, we manually evaluated the following five cases:

A. Failure to find login page,
B. not having reached status *submitted*,
C. detecting invalid credentials,
D. not reaching status *verified*,
E. reaching status *verified*.

We validate manually Shepherd's login procedure by creating five sets of 100 websites. In the first case, we manually visited sites, while for cases 2–5 we reviewed automatically created screenshots. In all cases, the evaluator was at liberty to skip sites containing adult content[5]. These cases give insight into performance of the heuristics and suggest which gains can still be made by improving heuristics.

A. Finding login pages

This case concerns websites where Shepherd was unable to find login areas. We evaluated 100 such sites and manually identified login areas on 44 sites. On not all of these, it was clear that the login area would provide a login for the initially visited site. One site was not manually evaluated, as it contained adult content. We did not discover login elements on the other 55 sites. Shepherd thus failed to identify login fields that were present on 44 out of 99 sites where it did not find login elements. Optimistically viewed, this can be generalised, which means that at most 9.5% more sites could be reached. For this dataset, that means that at best, $10,439 \cdot 44/99 = 4,640$ more sites could be included.

B. Submission failures

This case concerns sites where Shepherd found a login area and submitted credentials, but could not detect success.

[5]as such sites are more likely to contain illicit material.

We reviewed 100 screenshots of login areas of such sites. On 85 screenshots we derive that Shepherd found the correct login area. Another 8 show indications but raised uncertainties, as the login area was covered by a CAPTCHA or pop-ups in a different language. Another 5 cases were failures, where Shepherd focused elements belonging to registration elements (3) or ended up on age verification pages (2), instead of login elements (which were also present)[6]. Two cases could not be evaluated due to overlaid content or the site's language.

C. Detecting invalid credentials

This case concerns Shepherds ability to identify invalid credentials. We used the 100 screenshots from our second case 2. In 87 screenshot Shepherd correctly noted messages signalling invalid credentials (66) or the absence of such signals (21). In 8 cases Shepherd misclassified the responses from websites. The remaining 5 screenshots were not used, as these showed registration fields, were overlaid with pop-ups or not interpretable due to the language. We conclude that given a login area, the process of credential submission and evaluation of the success of that step performs reasonably well.

D. Submitted, but unverified

This case concerns sites where Shepherd successfully submitted credentials, but failed to verified it. Note that verification is supposed to fail when login was not successful. In 76 out of 100 cases, this was the case, underscoring the need to verify whether login is indeed successful. 21 cases showed clear signs that Shepherd entered the post-login stage, while 3 cases could not be verified. In other words: in at least 21% of the examined cases, this process resulted in a false negative. In the experiment, of the sites where credentials were successfully submitted, on 3,950 sites this could not be verified. 21% of this is 829 sites. These 829 sites are sites where logins were potentially successful, but not detected by Shepherd. Additional or improved verification methods thus may lead to hundreds of sites more evaluated.

E. Verified

This case concerns 100 sites which passed verification. Of these, only one site could not be checked. Two other screenshots showed that the user account was banned. Nevertheless, Shepherd was clearly logged in on these sites. For that, we find the verification process to have high accuracy ($\geq$97%) and therefore have high confidence in all findings on sites marked *verified*.

F. Performance of login-finding methods

Finally, we zoom in on the detection of the login area. In particular, we investigated the success rate of the different approaches to finding login elements. We remark once again that these methods are executed sequentially: only if methods 1–5 fail, method 6 is executed. Of the 38,431 domains where a login area was found, method 2 was most successful, finding more than 40% of login areas.

Method 1 - Landing page:	6,311
Method 2 - URLs (first level):	15,438
Method 3 - Clicking elements:	4,004
Method 4 - Try standard URLs:	3,745
Method 5 - Search Engines:	8,875
Method 6 - URLs (second level):	76

In conclusion, Shepherd managed to successfully login on 7,113 sites (submitted and verified). We found that the heuristics used in Shepherd an also be improved. Main areas for internal improvement are:

- improving identification of login elements.
  This can lead to (at most) 44% more sites reached, or 4,640 additional sites in this experiment.
- reducing false negatives for verification.
  This can lead to (at most) 21% sites where submission was successful, or up to 829 additional sites in this experiment.

However, the number one area for improvement is: valid credentials for more sites. Shepherd's detection of invalid credentials found that for 60.1% of sites where login elements were found, no valid credentials were available (in the experiment: 23,088 sites).

VII. POTENTIAL USE CASES

In this section, we highlight use cases for Shepherd. In general, we see two potential areas where Shepherd can boost security and privacy research. These are the investigation of post-login features across multiple websites, and the comparison of pre- vs. post-login aspects.

A. Measuring post-login features

By logging in, Shepherd receives authentication tokens from websites, which allows studying the security of such tokens. Shepherd possesses an implementation to extract authentication cookies, facilitating investigations into properties of these. Furthermore, Shepherd could be extended to identify sites whose login system exhibits specific behaviour. One example could be sites that store session identifiers in local storage instead of using cookies; another is looking for sites vulnerable so sub-session hijacking [CRB19] (which relies on presence of multiple authentication cookies).

Moreover, the automatic identification of authentication cookies enables construction of a ground truth for machine learning purposes (e.g., [CTBO14]). The most extensive set of authentication cookies reported in literature amounts to cookies from 215 sites (332 authentication cookies) [CTC+15]. Using the authentication cookie identification mechanism of Shepherd, we collected a set of authentication cookies for 6,335 sites – the automated mechanism failed to identify cookies on 778 out of the 7,113 sites where login was successfully verified.

Finally, Shepherd enables measuring adoption of security measures for logged-in users, such as cookie flags such as *SameSite*, *Secure*, etc.; HTTP-headers such as HSTS; CSRF-tokens [CCF+19]; etc.

[6]This sometimes happens when sites offer login elements outside of a form element. For such cases, other input elements can be confused with the login elements. Shepherd will use heuristics to select the most likely login related elements.

B. Anonymous visitors vs. logged-in users

The above suggestions for measurements can also be applied *before* logging in. Hereby, Shepherd enables studying the contrast between anonymous visitors and logged-in users on a large scale. For example, logged-in users may face less trackers than anonymous users. An additional direction is to use Shepherd to create authentic user profiles by gathering cookies from several sites. User profiles have already been used in earlier research to analyse ads [CMC+15] or evade bot prevention measures [SPK16]. Finally, Shepherd could be extended to automate logging out. This would enable a large-scale study comparing the session state while logged-in with the state after logging out, including e.g. identifying flaws in session invalidation.

VIII. Ethical considerations

For our study, we aimed to achieve an unprecedented scale in entering restricted areas of websites. As this led to concerns, we sought and received approval from our ethical review board. Nevertheless, we wish to highlight the various ethical concerns.

The primary concern was acquiring a large set of credentials from a legitimate source. Fortunately, the BugMeNot database is exactly this: a large set of login credentials with strict (and enforced) policies to ban a site from inclusion upon request of the site owner.

Secondly, the experiments must not exceed their mandate and break things. As Shepherd is designed to interact with websites as human do - To trigger elements human clicks are simulated, timeouts are used between each action, sites are not crawled in parallel, only visible elements are considered for interacting with a site - the risk of overburden or accidentally confusing the websites logic is reduced. We worked on this by testing Shepherd on a small number of domains and resolving any issues. The results are not 100% perfect, but the fraction of mistakenly pressed buttons we detected is very small.

Thirdly, the tools created can easily be misused, e.g., to apply credential stuffing or password guessing attacks. Therefore, we cannot and will not publicly release Shepherd. On the other hand, we welcome interest from fellow researchers. Thus, we will make Shepherd available for followup studies by other bona fide researchers upon request.

IX. Conclusions and Future work

Many previous works have studied the web. Most of these were limited to the public areas of websites. This implies that post-login aspects were hidden for such studies or could not be measured at scale. Research that attempted to address the post-login world, mostly fell back on manual intervention, to avoid the many challenges with an automatic approach [TDK11]. Only three previous studies had access to post-login areas of larger set of websites. All these used means to automate logging in, but were bound to certain type of logins. In this work, we took a generic approach to login on websites. For that, we designed and developed Shepherd, a tool that enables post-login measurements of unknown websites. As login processes are very diverse, automated logins cannot achieve full coverage. Moreover, the variety in login processes implies many design challenges. Shepherd accounts for this by several failure modes. The study conducted with Shepherd shows that large-scale automated login is feasible. Previous best efforts using a semi-automated approach [MFK16] managed 149 sites, while automated approaches reached 912 sites [ZE14] using single sign-on (SSO) credentials.

In contrast, Shepherd can use either domain-specific or SSO credentials. In a case study with domain-specific credentials, Shepherd achieved 7,113 successful logins, an order of magnitude beyond previous best effort at logging in automatically. In a limited experiment with Facebook SSO credentials, Shepherd achieved 383 successful logins.

Future work

We are redesigning Shepherd's SSO component to use a more generic approach for SSO logins, which will lead to supporting more single sign-on frameworks and make the SSO-related procedures more robust. Given that there are various SSO providers commonly used in non-Western countries, this enables various types of detailed comparison studies between countries. Secondly, we are planning to extend previous studies of cookie security for post-login cookies. Related to this, integrating Shepherd-alike capabilities into a privacy measurement framework such as OpenWPM would allow to study whether logged-in users gain or lose privacy compared to anonymous visitors. Similarly, a combination with an existing security scanner could allow remote security scans (e.g., SQL injection, XSS, CSRF) in the members-only area of websites.

Acknowledgment

The authors would like to thank the MADWEB'20 organisers for their travel support. In addition, we extend our thanks to Jun Pang, Olga Gadyatskaya, Agnieszka Jonker and Jana Polednovà for translating keywords. Finally, we are grateful for comments we received on earlier versions of this work from Marko van Eekelen and Greg Alpár.

References

[BCFK14] Michele Bugliesi, Stefano Calzavara, Riccardo Focardi, and Wilayat Khan. Automatic and robust client-side protection for cookie-based sessions. In *International Symposium on Engineering Secure Software and Systems*, pages 161–178. Springer, 2014.

[But10] Eric Butler. Firesheep. http://codebutler.com/firesheep/, 2010.

[CCF+19] Stefano Calzavara, Mauro Conti, Riccardo Focardi, Alvise Rabitti, and Gabriele Tolomei. Mitch: A machine learning approach to the black-box detection of CSRF vulnerabilities. In *Proc. 4th IEEE European Symposium on Security and Privacy (EuroS&P'19)*, pages 528–543. IEEE, 2019.

[CMC+15] Juan Miguel Carrascosa, Jakub Mikians, Ruben Cuevas, Vijay Erramilli, and Nikolaos Laoutaris. I always feel like somebody's watching me: *Measuring online behavioural advertising*. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies, CoNEXT 2015, Heidelberg, Germany, December 1-4, 2015*, pages 13:1–13:13, 2015.

[CRB19] Stefano Calzavara, Alvise Rabitti, and Michele Bugliesi. Sub-session hijacking on the web: Root causes and prevention. *Journal of Computer Security*, 27(2):233–257, 2019.

[CTBO14] Stefano Calzavara, Gabriele Tolomei, Michele Bugliesi, and Salvatore Orlando. Quite a mess in my cookie jar! Leveraging machine learning to protect web authentication. *In Proceedings of the 23rd international conference on World wide web*, pages 189–200, 2014.

[CTC+15] Stefano Calzavara, Gabriele Tolomei, Andrea Casini, Michele Bugliesi, and Salvatore Orlando. A supervised learning approach to protect client authentication on the web. *TWEB*, 9(3):15:1–15:30, 2015.

[dRND+12] Philippe de Ryck, Nick Nikiforakis, Lieven Desmet, Frank Piessens, and Wouter Joosen. Serene: Self-reliant client-side protection against session fixation. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 59–72. Springer, 2012.

[EN16] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1388–1401. ACM, 2016.

[GRC+18] Mohammad Ghasemisharif, Amrutha Ramesh, Stephen Checkoway, Chris Kanich, and Jason 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)*, pages 1475–1492. USENIX Association, 2018.

[JKV19] Hugo Jonker, Benjamin Krumnow, and Gabry Vlot. Fingerprint surface-based detection of web bot detectors. In *Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part II*, pages 586–605, 2019.

[MFK16] Yogesh Mundada, Nick Feamster, and Balachander Krishnamurthy. Half-Baked Cookies: Hardening cookie-based authentication for the modern web. In *Proc. 11th Asia Conference on Computer and Communications Security (ASIACCS)*, pages 675–685, 2016.

[NMY+11] Nick Nikiforakis, Wannes Meert, Yves Younan, Martin Johns, and Wouter Joosen. Sessionshield: Lightweight protection against session hijacking. In *International Symposium on Engineering Secure Software and Systems*, pages 87–100. Springer, 2011.

[RB14] Nicky Robinson and Joseph Bonneau. Cognitive disconnect: Understanding facebook connect login permissions. In *Proceedings of the second ACM conference on Online social networks*, pages 247–258. ACM, 2014.

[SPK16] Suphannee Sivakorn, Iasonas Polakis, and Angelos D. Keromytis. I am robot: (deep) learning to break semantic image captchas. In *Proc. 1st IEEE European Symposium on Security and Privacy (EuroS&P'16)*, pages 388–403. IEEE, 2016.

[TDK11] Shuo Tang, Nathan Dautenhahn, and Samuel T King. Fortifying web-based applications automatically. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 615–626. ACM, 2011.

[vAHS17] Steven van Acker, Daniel Hausknecht, and Andrei Sabelfeld. Measuring login webpage security. *The 32nd ACM SIGAPP Symposium On Applied Computing*, 2017.

[ZE14] Yuchen Zhou and David Evans. Ssoscan: Automated testing of web applications for single sign-on vulnerabilities. In *Proc. 23rd USENIX Security Symposium*, pages 495–510. USENIX Association, 2014.

Appendix

A. Facebook extension

In addition to domain-specific credentials, we build an extension to support single sign-on credentials. The main benefit of using SSO credentials is that this avoids the need to acquire a large set of valid credentials. On the face of it, it would seem that adapting Shepherd's core functions to login with SSO would be straightforward. In practice, we encountered difficulties doing this. As a result, we designed a further set of functions that allow Shepherd to handle the specific nature of logging in with SSO intermediaries. The following focusses on deviations from the domain-specific login design.

a) Searching for SSO login buttons: SSO login areas are usually activated by triggering an interactive element. Hence, the login area cannot be identified by searching for standard login elements such as a field for username or a link labelled 'login'. Rather, SSO-specific elements must be identified. Unfortunately, these elements are not standardised between SSO intermediaries and often differ between websites, even for one intermediary. An additional challenge is that some SSO intermediaries also provide social media features, such as a 'like' or 'share' button. Distinguishing such elements from the sought-for login elements is difficult. Similar to Zhou and Evans, we address this by filtering interactive elements based on a set of keywords. These keywords are also specific to the SSO intermediary, which poses an additional challenge for supporting multiple intermediaries. We also encountered that in a case of insufficient filtering, these elements can lead to false positives as these produce similar login dialogues. We avoided these by adding additional checks for URLs.

b) Submitting and verification: A hurdle is that after logging in with single sign-on, the website requests that the user fills in an enrolment form of some kind. Access to other parts of the site is blocked until this form is filled in. This behaviour does not always occur, but frequently enough that it affects the success rate. While that presents an obstacle to achieving the full flexibility that domain-specific credentials offer, it does not impede gauging the usefulness of single sign-on for enhancing the coverage of Shepherd. For that, forms do not have to be filled in. Unlike domain-specific credentials, SSO credentials can hold additional details (e.g., real name or associated phone number), which can be used in future versions for improved verification procedures or to finish enrolment.

c) Implementation and performance: We developed an initial extension to Shepherd's core functions to login using Facebook's SSO service. In our testing set of 50 sites containing SSO logins for Facebook, it was able to identify login areas on 41 sites. Shepherd missed login areas on 7 sites, while two scans failed. Once Shepherd has found an SSO login, it processes the login as a regular login, but with the supplied Facebook credentials. The current identification process for Facebook logins is based on domain filtering (of URLs in the visited page). While this proved to be effective in practice, this does imply a certain amount of fine-tuning is needed to support additional SSO providers.

Due to the above mentioned difficulties in identifying the correct elements for SSO logins, the extension is significantly slower than regular Shepherd. The extension is able to scan about 3,000 sites per machine per day.

B. Validation of the Single Sign-On extension

We scanned the Alexa Top 10,000 sites using single sign-on credentials to validate the extension. We remark here that these sites do not all support single sign-on with Facebook credentials. This is intentional, as Shepherd should be able to scan any site. The scan was carried out with two machines, each using its own Facebook account, created newly for this purpose. We further divided the target domains into 4 equal sets, so that the results can be examined between scanning these sets. The first machine was used to scan the first 3

	# sites	out of	
Total	10,000	–	
Sites reached	8,829	10,000	(88.3%)
SSO login found	2,057	8,829	(23.3%)
Submitted	1,915	2,057	(93.1%)
Verified	383	1,915	(20.0%)
Auth cookies found	330	383	(86.2%)

TABLE IV: Performance of the Single Sign-On scan

sets, while the second machine scanned only the last set. The Facebook account for the first machine was blocked at a certain point while scanning the second set, due to posting inappropriate content. This was caused by Shepherd clicking share buttons on visited sites (of an adult nature). We adjusted the extension to address this by blacklisting certain types of Facebook URLs. After recovering the blocked account and scanning the third set, we found 55 shared posts on the account (each of which must be due to a successful login). Shepherd misclassified these as logins. The second Facebook account was also blocked, this time due to 'suspicious behaviour'. Recovery of this account was more involved and therefore omitted. Unlike the other 3 sets, we thus could not verify Shepherd's results for this set in the Facebook account.

As shown in Table IV, Shepherd thought it recognised Facebook-based SSO logins on around 20% of the Alexa Top 10K. While Shepherd was able to submit credentials to 93.1% of these, verification only succeeded on 20% of the *submitted* sites. It could be that Shepherd's default verification process is unsuitable for SSO logins, or, perhaps, often an additional registration form appeared and full site access was not yet granted. Shepherd set the percentage of sites on which it believed to have found Facebook Login at 20%, and the percentage of sites where it successfully verified login at about 4%. Logging in with Facebook leaves traces in the permission settings of a user account. We checked this setting four our first account and found 664 apps with specific permissions. To compare these numbers, we looked for reliable numbers on the adoption of Facebook Login. The reported rates we encountered predicted significantly lower adoption. Nevertheless, we believe that the SSO login detection algorithm can be further improved to reduce false positives.