Formguard: Continuous Privacy Testing for Websites Using Automated Interaction Replay

Tim Vlummens
COSIC, KU Leuven and Radboud University
Leuven, Belgium and Nijmegen, The Netherlands
tim.vlummens@esat.kuleuven.be

Gunes Acar
Radboud University
Nijmegen, The Netherlands
g.acar@cs.ru.nl

***Abstract*—Websites commonly use third-party scripts for purposes such as advertising, analytics and payment processing. In recent years, several popular third-party scripts fell victim to supply-chain attacks where users' login credentials and credit card details were stolen. These devastating attacks sometimes remain hidden for several weeks until they are discovered. In this paper, we present Formguard, a continuous testing tool that detects web-based supply-chain attacks in an automated manner. Formguard allows website owners to record complex interactions such as logging in, signing up or checking out a product on their websites. These recordings can then be periodically replayed, while monitoring the HTTP requests and WebSocket messages, accesses to input fields, and information on the embedded scripts. The periodic and automated testing allows for faster detection of malicious supply-chain attacks and potential compliance issues that are impossible to detect with non-interactive security scanners. While Formguard specializes in detecting digital skimming attacks, it can also perform various privacy tests against different aspects of a website including embedded scripts, HTTP headers and cookies. We evaluate Formguard through two case studies. First, a long-term robustness test on 75 websites shows that even complex recordings remain replayable for several months, suggesting minimal maintenance workload for website owners. Second, we use Formguard's crawl mode to study access to and exfiltration from login and registration forms on 100,000 websites, revealing access to password fields on over 10K sites by third-party scripts. Finally, we discuss the challenges of automated testing for modern web forms, providing insights that may benefit researchers and practitioners.**

***Index Terms*—privacy, security, supply-chain attacks, data exfiltration, testing**

1. Introduction

The majority of websites today make use of third party scripts to implement a variety of functions. These may range from including advertisements to providing payment forms for online shopping. As some of these third party scripts are used by large numbers of websites, they become an attractive target for attackers to perform so called supply-chain attacks. In these attacks, malicious actors inject their malicious code to a common third-party script, for example, by compromising their server. When a user visits a site that embeds the compromised script, the malicious code could redirect the users to unwanted sites or steal their credentials and credit card details. A recent example of this is the polyfill.io attack, where the polyfill.io domain was bought by a Chinese company in February of 2024 [1]. In June of the same year, it was warned that the company had changed the script to injected malicious code on sites implementing the cdn.polyfill.io scripts, redirecting users who visited a site with the compromised script to unwanted other sites. While the polyfill.io attack was quickly detected, other supply-chain attacks on high-profile websites such as British Airways have remained undetected for weeks [4]. The high impact these attacks can cause, alongside the time it takes to notice them, indicates the need for an automated detection tool which can be used by website owners to test their site on a regular basis.

Prior research on exfiltrations of personal data, such as passwords, email and credit card details, generally focus on large scale web measurements [2], [26], [28]. As the crawlers used for these studies need to work on a large number and variety of sites, they prefer generic detection methods to find and fill different form fields, either on the landing page or some of the inner pages. However, pages, forms or content that are served only after complex interactions, such as booking a flight or opening an account, may not be analyzed by such generalist crawlers. These *blindspots*, however, are more likely to contain sensitive form fields such as credit card details on a payment screen.

In this paper, we present Formguard, a tool to detect web-based supply chain attacks through automated tests. Formguard allows a user to record their interactions on a webpage, such as navigating to and filling login or payment forms, including in shadow DOM. These interactions can then be replayed, allowing for regular, repeated testing. The ability to record and replay interactions ensures that a website owner is able to test desired flows or form fields specific to their website. During the replaying of the recorded interaction, HTTP requests and WebSocket messages, HTMLInputElement API accesses and information on embedded scripts, such as the content hash and the stacktrace, are saved. The saved data can then be analyzed to check for unexpected script behavior, data exfiltrations via HTTP requests or the WebSocket API, as well as for other privacy tests set up by the website owner.

In addition, Formguard can be used to perform large-scale automated web crawls. In the crawl mode, Formguard uses pre-trained machine learning (ML) models to identify login and registration pages, and it detects and

fills possible login forms. We use this crawl mode to perform a crawl on 100,000 popular websites, covering both home and inner pages. This large-scale exercise ensures that Formguard can capture the data required to identify credential exfiltrations from web forms.

In particular, we make the following contributions:

1) We develop Formguard, a continuous privacy testing tool for website owners with a focus on detecting digital skimming attacks.
2) We evaluate the long-term maintainability of Formguard's recordings by recording and replaying interactions on 75 websites. These recordings are replayed over three months to assess the necessary upkeep efforts.
3) We test Formguard's robustness by crawling 100,000 websites where we attempt to detect digital skimming attacks.

2. Related Work

Starov et al. reported on the first large-study of the leakage of information filled into contact forms on the 100,000 most popular websites [28]. Their crawler filled in contact forms found on the visited websites and checked the recorded HTTP requests for leakage of the filled values. They found that 6.1% of all submitted contact forms leaked information to third parties. In addition, Starov et al. developed Formlock, a browser extension that warns and protect visitors against forms that leak personal information. Unlike Formlock, Formguard can be used for more general privacy tests and it is aimed at website owners, allowing for automated testing for leaks as a result of interactions with a website.

Senol et al. presented a study of the misuse of access to online forms by online trackers [26]. From two vantage points, they measured the email and password collection before submission of online forms on the top 100,000 websites. They found that 1,844 websites in the EU crawl and 2,950 websites in the US crawl exfiltrate emails to tracker domains, without the consent of the user. In addition they found that roughly 50 websites leak users' passwords to tracker domains before form submission. The password leaks were largely fixed due to their disclosures.

Acar et al. studied the data exfiltration by third-party scripts directly embedded on web pages [2]. They found the use of invasive practices by third-party scripts, such as the insertion of invisible login forms onto a page, which in turn trigger the login autofill of a browser. This allows the third-party script to read out the auto-filled in email address, sending its hashes to data broker or other third-party domains. In addition, they found the leaks of email, password, credit card and health data initiated by session replay scripts, as they exfiltrate the whole Document Object Model (DOM) to reconstruct users' interaction with the page.

Nikiforakis et al. investigated the inclusion of JavaScript libraries on the top 10,000 Alexa sites and developed a set of metrics to grade the maintenance quality of the providers [22]. They show that some of the top ranking sites include scripts from providers with a low maintenance score. These providers could be potential *weak spots* and leave the sites implementing them open to attacks. In addition, they also describe four types of vulnerabilities related to the unsafe inclusion of JavaScript.

Van Acker et al. found that 51.3% of the Alexa top 100,000 domains contain a login page, with 70.2% of those pages embed third-party resources that can access password fields and are implicitly trusted [30]. Different than our work, Van Acker et al. focus on quantifying compromise risks under various adversary models such as an active network attacker capable of intercepting and decrypting the traffic to steal users' credentials.

Bouhoula et al. presented the first large-scale automated analysis of cookie notice compliance, enabled by ML models they developed [5]. They report that despite explicit consent rejection, 65.4% of websites set Analytics/Advertising related cookies and likely continue to process personal data.

While Formguard can be used to perform large scale web measurements on form leaks (see Section 5) or third-party inclusions, it is primarily designed as an automated privacy and security testing tool for website owners. Formguard allows recording of interactions specific to a website, which can then be replayed periodically. Through built-in network monitoring and JavaScript instrumentation, Formguard can detect security and privacy issues including supply-chain attacks targeting sensitive form fields.

Other privacy testing tools have been developed previously. Drakonakis et al. created `ERNIE`, a browser extension which can be used to visualize six cookie-based tracking techniques [31]. The authors discovered at least one form of tracking on 62% out of 385 health related websites before interacting with a consent dialog. Wesselkamp et al. developed a automated black-box auditing framework to analyze web-apps for their susceptibility to various cookie-hijacking attacks and their security mechanisms [13]. Contrary to the previous tools, Formguard is aimed at the detection of filled information in network requests. However, it also captures accesses to `Document.cookie` getters and setters, allowing owners to define their own tests on these.

In a 2008 study Jung et al. designed Privacy Oracle, a system capable of analyzing the network traffic of installed desktop applications for leaks of user information [19]. They discover leaks by linking changes in user input to changes in the network traffic. Testing 26 applications, such as media players and instant messaging client, Privacy Oracle discovered previously undisclosed information leaks, including cases where identifying information is regularly sent in clear text. In contrast to Privacy Oracle, Formguard is aimed at discovering leaks on websites. Leaks are detected by attempting to find encrypted and decrypted versions of the filled information in the intercepted requests.

3. Formguard Record-Replay

To allow for repeated testing for detected leaks, Formguard offers the option to record and replay interactions with a page. In contrast to automated navigation, this allows Formguard to perform more difficult navigation flows, such as a checkout procedure.

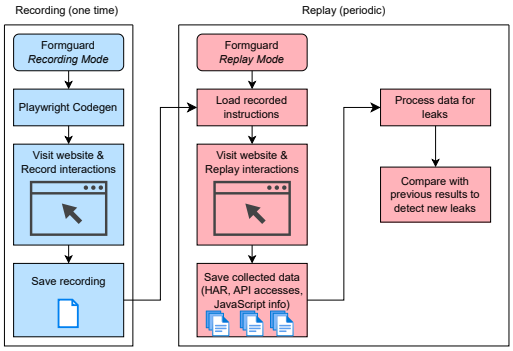

Figure 1. High level overview of Formguard's record and replay mode

Formguard uses Playwright [24] to navigate to a site, record the interactions made by a user and later replay the recorded interactions. Playwright is a Python library built for reliable end-to-end testing and can be used as a general purpose browser automation tool. It offers utilities to capture the network traffic and supports Chrome Devtools Protocol (CDP) [6], which offers low-level access to the browser under test.

While replaying the previously recorded interactions, Formguard monitors the network requests and access to input elements by overwriting the getters of the `HTMLInputElement`. The data collected during replay can then be used to check for potential skimming attacks or test various privacy features defined by the website owner. For instance, the website owner may verify that their website sends particular HTTP headers, sets specific cookies and embeds particular scripts. They can check the integrity of the parsed scripts for signs of possible compromise thanks to the deep integration of Formguard with the JavaScript engine via CDP. A high level overview of the replay and recording steps can be found in Figure 1.

3.1. Record Mode

In recording mode, Formguard opens an empty browser window and a Playwright Inspector window [12]. After starting the recording, all manual interactions such as loading a page, clicking a link and filling form fields are automatically converted to equivalent Python instructions using Playwright's codegen feature [25]. After all desired interactions are made with the website, the automatically generated code can be saved to a file for later replay. By default, after a recording has been made, Formguard will automatically switch to replay mode and replay the new recording. This allows the user to immediately verify that the new recording functions correctly.

3.2. Replay Mode

In order to replay a recording, the user invokes Formguard with the recording to be replayed. This opens a fresh browser window and the specified file with recorded code is replayed instruction-by-instruction. Network requests made during a replay, which are used for detecting exfiltrations from form fields, are saved into a HAR file, a JSON-formatted file to store HTTP network requests. Accesses to the cookies and input fields are captured and stored to a JSON file by overwriting the getters and setters of the `HTMLInputElement` and `Document.cookie`. In addition, embedded script information is collected by handling the CDP's `Debugger.scriptParsed` event [11]. This event is fired when the JavaScript engine parses a script, including dynamically executed scripts through `eval` or the `Function()` constructor [8]. The information exposed in the `scriptParsed` event object includes the script's URL, stack trace, content hash and embedding context. In addition to storing these details about each parsed script, Formguard also stores timestamps for filling the form fields and the values entered in each field, as well as the WebSocket messages made during the visit. After each replay, the recorded data can be compared against the baseline of the first replay to check for unexpected leaks. For convenience, Formguard also accepts a folder containing multiple recordings, automatically replaying them one after another.

Handling closed Shadow DOM. Websites may use Shadow DOM to encapsulate custom elements from CSS and JavaScript. While this isolation carries benefits for the website, it may render it impossible to monitor them for leaks with Formguard, especially for elements in the Shadow DOM using the "closed" mode, as Playwright will be unable to record and replay interactions on them [23]. In order to deal with this challenge, upon navigating to a page, Formguard injects a script before the page scripts are run. The injected script overwrites the `attachShadow` method [3] used when attaching a shadow DOM to replace its mode parameter so that it always attach the Shadow DOM in "open" mode instead. This enables access to and monitoring of the forms in Shadow DOM, both in record-replay mode and in the crawls. We adopt this method from a preliminary study by de Vries [10].

To make the replays more robust against website dynamism and non-determinism, two extra features are added to Formguard.

Handling optional elements. Each visit to a webpage is influenced by a myriad of factors, including time, vantage point or website dynamism. For instance, some websites may display a popup dialog only in some visits based on a fixed probability. If an element is displayed and interacted (e.g., closed) during a recording and not during the replay, the replay may fail due to the missing but expected dialog element. In order to handle this dynamism, Formguard allows the user to mark certain instructions in a recording as optional by appending `#OPTIONAL` to the end of the line. If an optional instruction fails to execute because the element is not found, the replay continues normally, and subsequent instructions are executed without marking the replay as failed.

Fuzzy frame matching. The second feature was added to make Formguard robust in cases that involve iframes with random `id` or `name` attributes. An example of such frames is those provided by Stripe [29] for online payment forms. These frames have variable `name` attributes that follow the format `__privateStripeFrameXXXX`, where the last four characters are numbers that change in each visit. As instructions generated by Playwright's codegen will often refer to frames using their `name` attribute, form elements in these frames cannot be accessed. To mitigate this issue,

Formguard offers a fuzzy frame matching option. When the option is enabled and an instruction containing a frame locator fails, the other frames on the site will be checked to see if they match the original frame. The features preferred by Playwright's codegen to identify elements inside the frames (e.g. placeholder text, associated label), generally do not change when the name of the frame changes. Thus, Formguard uses the element locator to check if it can be found in other frames on the same site. If such a frame is found, its locator is saved, enabling subsequent instructions to immediately use the correct new frame.

4. Long-term Robustness Test

4.1. Test Setup

To test the long-term durability of Formguard's recordings, we used Formguard to record our manual interactions on a total of 75 websites: 60 websites with login forms and 15 with donation forms. The recordings for the 60 websites with login forms involved navigating from the landing page to a login form, filling and submitting the form. The recordings for the 15 sites with donation forms involved navigating from the landing page to the donation form and filling and submitting it. These recordings are then replayed after different durations to check what percentage of the recordings still replay without failure. We chose websites with login and donations because they often involve sensitive user data such as passwords or credit card details.

The 60 websites with login forms are sampled from the top 100,000 websites as determined by the CrUX list of October 2024 [9]. We took 20 sites from each of the top 1,000, top 10,000 and top 100,000 sites, excluding the previous range. Each site was manually visited to check for the presence of a login form, and replaced by a new site from the same popularity bin if no login form was found. For these recordings, no instructions are marked as optional and fuzzy frame matching is not enabled.

The 15 donation sites are sampled from a list of 3,081 sites that were predicted to have a donation form. This list was inherited from a preliminary study by Kokkelmans [20] and was constructed by searching for `intext:"Cardholder" AND inurl:"Donate"` on Google and other search engines. For these recordings, no instructions were marked as optional, but fuzzy frame matching was enabled because 11 out of the 15 sites had parts of the donation form in an iframe with a name that changed with each visit.

The recording and replaying of the 75 recordings were performed from a European Union country from October 2024 to January 2025 using a residential connection. The 60 login recordings were replayed weekly for five weeks, followed by a final replay after 13 weeks. The 15 donation recordings were run once after one week and once after 14 weeks of their recording. Thus, for both website categories, the time between the first and the last replay was more than three months. The mismatch in the testing frequencies was largely due to logistical constraints, which we accept as a study limitation.

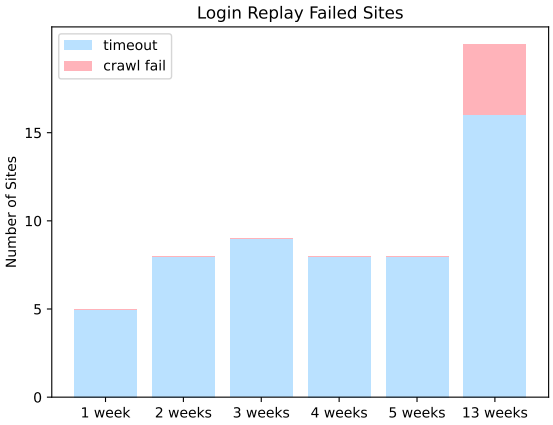

Figure 2. Results from the robustness test of the replay function for login pages

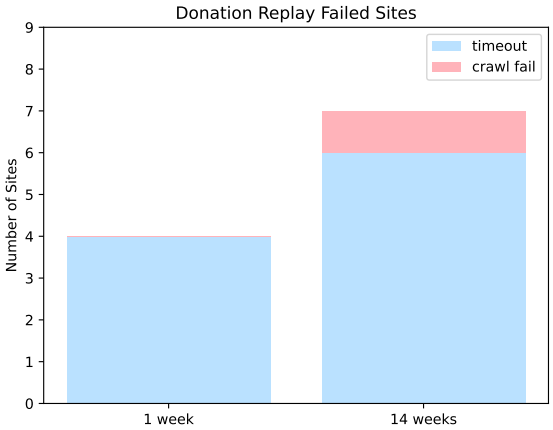

Figure 3. Results from the robustness test of the replay function for donation pages

4.2. Results

After five weeks, eight out of the 60 login recordings have at least one instruction time out. After 13 weeks, the number of failed sites have increased to 16/60 sites. For the donation pages, 4/15 failed after one week, and 6/15 failed after 14 weeks. The results for both the login and donation recordings can be seen in Figure 2 and Figure 3, respectively. The 4/60 sites and 1/15 sites with a crawl failure were caused by either a page not loading or a problem in crawler logic which has since been resolved.

The timed-out instructions are due to two reasons. The first reason involves the appearance of dialog windows or screen blocking windows. If such windows load later than expected, the instruction to dismiss them may time out and potentially prevent subsequent steps from executing. The second reason occurs due to the changing of page elements after recording. If an element the recording interacts with changes its recorded identifier, the replay will not find that element. Important to note is that these results reflect a worst-case scenario. None of the instructions interacting with dialog windows in the test recordings are marked as optional. Marking these as optional prevents the varying load times of the windows causing timeouts on their own.

Overall, the results of our long-term tests suggest that Formguard recordings can remain reliable over an

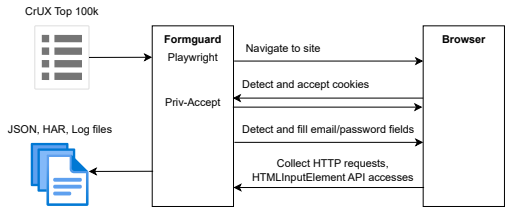

Figure 4. High level overview of the automated crawler portion of Formguard

extended periods of time. They do not require frequent re-recordings and can be replayed for months without any breakage — reducing the maintenance costs for website owners.

5. Automated Web Measurements

In addition to recording and replaying interactions on specific websites, Formguard can be used to perform crawls to study privacy issues related to online forms at scale. This mode also allows the testing of the data collection methods used for record-replay on a larger number of websites. In the automated crawl mode, Formguard searches for and fills login or registration forms on multiple websites. When all form fields on a page are filled, Formguard submits the form. Similar to record-replay mode, HTTP requests, WebSocket messages and access to input elements are monitored to detect potential leaks, and closed shadow DOM's are converted to open (Section 3.2). Moreover, the `pyvirtualdisplay` library is used to run the crawler in headful mode to avoid bot detection. The interactions made on sites had not been previously recorded. Instead the crawler used a generalized method to detect the required elements. A high level overview of the crawler can be seen in Figure 4.

5.1. Consent Dialogs

More websites started to show cookie consent banners after the introduction of GDPR in 2018 [17]. To increase the likelihood of a leaking script being active, the crawler accepts all forms of personal data processing using a consent interaction module we build by porting Priv-Accept from Selenium to Playwright [18] [21]. Priv-Accept works by searching for HTML elements that can contain a consent option, such as <button>, <a> and elements. It then checks the text content of these elements against a list of keywords and phrases such as "Accept" or "Agree & continue".

5.2. Detecting Login Forms

Formguard's `mode` and `amount` command line parameters let the user specify *where* the crawler searches for email and password fields. In all modes, the crawler first searches for these fields on the landing page. In addition, it can also look for these fields on a number of inner pages. For identifying if a page is a login or signup page, the crawler uses a machine learning-based classifier adapted from a 2024 study by Senol et al. [27]. The classifier uses 88 distinct features, such as the inclusion of "login" or "sign-up" terms in button texts or the presence of a "Remember Me" checkbox element, to determine whether or not the page is an authentication page.

When a page is determined to be a login or signup page, the crawler searches the page and its frames for password and email fields. For detecting password fields, the crawler takes all input fields with the `password` type (i.e. `input[type='password']`). However, email fields are not required to have the `email` type (i.e. `input[type='email']`). In addition to inputs with the `email` type, the crawler also looks for text inputs or inputs without a type that fulfill certain rules. These rules are based on a pretrained email field classifier from Mozilla Fathom [14]. Fathom is a supervised-learning system for recognizing parts of web pages such as pop-ups [16]. The usage of the email classifier model normally requires the injection of a large script into the page's global context, causing name collisions and other errors in certain cases. To prevent this injection, the crawler uses a rule-based method that relies on the model's features, but not its weights. It checks all input fields for the identifying features used in the ruleset of the model (e.g., associated label, placeholder text, name,... containing 'mail' or 'email' keywords). Instead of then adding weights to each rule, the crawler assumes the input field to be an email field if it fulfills at least one of the rules.

The simplified detection method is tested against the original classifier on the top 1000 sites as determined by the CrUX list [9] (as of July 2024). Of these, 302 sites contained a login or registration form on its landing page or one of its inner pages. This resulted in 654 total pages containing a login or registration form. Table 1 shows the results of the comparison between the two methods. Note that the page amounts do not sum to the total number of pages, as a single page can contain both an email field with and an email field without the `email` type. On three pages, the simplified model correctly detected more email fields than the original classifier. On only one page did the simplified missed an email field, due to Playwright being unable to link the label to the corresponding input field. Following these results, it was decided that the simplified model was sufficiently accurate for the detection. The 147 sites where no email fields were detected by both approaches had either a different identification field (e.g. telephone number, username,...) (85 pages), were in a non-roman alphabet (46 pages) or had a unique reason for not being detected (16 pages).

5.3. Filling Forms

Once the crawler has found email and password fields on a login or sign-up page, it will attempt to fill them. To combat potential bot detection, the detected input fields are filled by simulating a human-like typing behaviour. The mouse moves over to and clicks the input field before filling it character-by-character, with randomized intervals and dwell times for each keypress and click. In addition, the tab key is pressed after filling each field. Once all fields are filled on a page, the crawler tries to submit the form of the last field it filled. It does this by trying to access the field's `form` attribute and using the `form.submit` function. The timestamps for each filled field and submitted

TABLE 1. COMPARISON SIMPLIFIED MODEL AGAINST ORIGINAL FATHOM CLASSIFIER

Category	#Sites
Successfully visited	950
Login or registration forms detected	302
Category	#Pages
Login or registration form detected	654
‘”input=[type=’email’]”‘ fields	283
Equal email detection between both models	229
Fewer email fields detected by simplified model	1
More email fields detected by simplified model	3
No email fields detected by both models	147

TABLE 2. CRAWL STATISTICS

Category	Number of sites
Successfully visited	93,228
Website did not load	6,576
Time out	169
Crawl failure	27
Interacted with consent dialog	19,163
At least one field filled	28,107
Sites with successful submit	27,113
At least one email field filled	18,830
At least one password field filled	26,116

form are saved alongside the other collected information. We comment on an unexpected side effect of using the `form.submit` function in Section 6.2.

5.4. Leak Detection

In addition to being sent unencoded in a request, the information filled on a web page can also be exfiltrated encoded, hashed or obfuscated forms. This could be done by third-parties to hide that they are collecting this information. Thus, when searching for form field exfiltrations, encodings have to be checked for these possibilities. For this purpose, we used a modified version of Englehardt et al.’s leak detector [15]. The leak detector works by creating a precomputed pool that contains all possible sets of searched information (e.g., filled password) by iteratively applying the hashes and encodings to the values filled on a page. The detector then checks the POST body, URL, referrer header and cookies of each requests. This is done by splitting it on potential separator characters, such as “=” and applying possible decodings to the seperated values. The repeated encodings and decodings are checked against the precomputed pool until a depth of three layers is reached.

6. Automated Web Measurement Results

6.1. Crawl Configuration

Using the crawler described in the previous section, we crawl the top 100,000 websites from the CrUX list [9] of October 2024. The crawl is run from the 14th to the 17th of November from the EU (Frankfurt) on a cloud-based DigitalOcean server equipped with 16 cores and 32GB RAM. The filling of fields is limited to four input fields per page, with a maximum of three filled pages per site. The maximum crawl duration is limited to 300 seconds per site with a maximum page load time of 30 seconds. These crawl parameters are determined using a test crawl on 1000 sites.

6.2. Results

The crawler visited 93,228 of the 100,000 sites successfully, taking on average 21.9 seconds per successful site, including load times. Of these sites, 18,830 had at least one email field filled, while 26,116 had at least one password field filled. The average visit time for sites with at least one field filled is 45.72 seconds. Visits to 27 sites were failed due to a crawler-related error. The high-level statistics related to the crawl can be found in Table 2. The crawler successfully submitted a form on 27,113 sites, with an average of 1.6 forms submitted per site. This shows that not all forms are successfully submitted on a site, due to the used method not finding the form attribute of the last filled element. An average of 3.0 fields are filled on each site, with 1.2 being email fields and 1.8 being password fields.The higher count of filled password fields is likely because most login and signup forms include a password field, whereas email fields may be replaced by a username or phone number.

The collected data shows that a third-party script reads the contents of an filled email or password field on 9,320 and 10,403 sites, respectively. The most prevalent script domains that access these input fields are given in Table 3. It is important to note that these accesses to the input fields may have legitimate reasons. For example, the script from the `auth.fandom.com` domain, while being a third party, appears to be used for authentication. Further, access to a field may not necessarily result in exfiltration. For instance, session replay scripts such as Microsoft Clarity (`www.clarity.ms`) may read the input fields to count the number of characters and display them in redacted form in the reconstructed videos of the session.

The results of detected email and password leaks are not shared in this paper, as they do not represent users’ experiences due to an unexpected interaction between the markup of the forms and how our crawler submits forms. In the initial analysis, we found the number of detected password leaks to third parties was unexpectedly high (3,532 websites). Through manual review on a subset of the detected leaks, we found that submitting forms by invoking the `form.submit` method causes the filled in email and password values to be reflected in the URL on some websites, as if the form’s action was `GET` [7]. This causes further leaks in the Referer [sic.] header of subsequent requests. However, these types of leaks would not be present when a real user interacts with the page — which we verified through manual tests. While this limitation in our method reduced the usefulness of our crawl data for detecting exfiltrations, it also provided a valuable lesson on best practices in studying online forms at scale. Furthermore, the crawl demonstrated Formguard’s ability to navigate websites effectively, detect login and signup pages at scale using its built-in ML models. Formguard also successfully filled close to 84,500 email and password fields on over 28,000 websites, interacting with diverse

TABLE 3. MOST PREVALENT SCRIPT DOMAINS BY NUMBER OF DISTINCT SITES FOR ACCESS TO PASSWORD AND EMAIL FIELDS. THESE ACCESSES MAY NOT NECESSARILY INDICATE AN ATTEMPT TO EXFILTRATE THE DATA.

Domain: password access	Number of sites	**Domain: email access**	Number of sites
www.gstatic.com	2,656	www.gstatic.com	1,905
www.clarity.ms	1,569	www.clarity.ms	1,454
www.googletagmanager.com	1,420	www.googletagmanager.com	1,418
ajax.googleapis.com	278	s.pinimg.com	454
auth.fandom.com	276	auth.fandom.com	270
static-ah.xhcdn.com	261	static-ah.xhcdn.com	261
static.xx.fbcdn.net	241	static.xx.fbcdn.net	246
mc.yandex.ru	208	ajax.googleapis.com	221
www.craigslist.org	156	cdn.shopify.com	209
t.contentsquare.net	140	mc.yandex.ru	168

and complex page designs while causing a failure on just 27 sites.

7. Discussion

In this paper we show that Formguard could be used in periodic privacy tests to uncover personal information leaks. Possible supply-chain attacks can be identified by detecting deviations in the expected access and transmission patterns of personal data. Our limited record-replay robustness tests show that Formguard can replay complex interactions for several months for most websites. This demonstrates that using Formguard for periodic tests will require little to no maintenance from the site owners. The ability to mark certain replay instructions as optional prevents replay failures due to web page non-determinism. Unlike our robustness tests, the site owners will also know when the layout of their site changes and can update the recordings accordingly.

The automated large-scale crawl shows that Formguard can be used to efficiently search for, detect and fill in online forms at scale. The issues with the detected leaks for the automated crawl highlight the importance of sanity checking automated crawl data. Instead of trying to find a submit button, the crawler used the `form.submit` method of the last filled elemen. This approach triggered accidental leaks of the filled values on certain websites, which would not have been encountered if submit button was clicked. We plan to adopt a form submission method that better reflects how users submit forms — by clicking the submit button. In addition to the email and password fields, the automated crawl mode could be expanded to detect and fill other fields, such as username and telephone fields.

We do not present Formguard as a compliance checking tool, because that requires more in-depth knowledge of the business and data processing practices of the website under test. Formguard's main use is allowing the recording and replaying of realistic interaction with a page, and the detection of leaks during these replays. The recorded data of a visit can additionally be used by the website owner for additional checks, such as checking cookies and HTTP headers.

8. Limitations

As with all automated measurements, there is risk that our crawler is detected as a bot and treated differently. We tried to prevent this as much as possible by adding human like behaviour to clicking and typing and running the crawler in headful mode within a virtual display.

As mentioned previously, the form submission method used in the crawl leads to non-representative results, but this does not impact the detection of leaks in the replays. In the replay mode, forms are submitted exactly how they were submitted during the recording — likely by clicking the submit button.

Our web measurements are taken from a vantage point in the EU, and involved giving consent to all personal data processing and cookies. We plan to expand our measurements to other vantage points and consent modes (e.g. Reject).

9. Conclusion

We presented Formguard, a tool that allows website owners to perform periodic privacy tests involving complex flows on their websites. Using Playwright's codegen functionality, Formguard allows website owners to record and replay complex flows, while monitoring for unexpected data exfiltrations, among others. Thanks to its low-level access to the JavaScript engine via the Chrome DevTools Protocol, Formguard can also be used to detect supply chain attacks involving compromised third-party scripts.

Through a long-term robustness test that lasted more than three months, we showed that Formguard recordings can function for several months without any maintenance. Features such as fuzzy frame matching and optional instructions provide greater resilience against replay breakage due to website changes. Finally, our large scale web crawl shows that Formguard can reliably discover and fill in forms at scale using ML models adopted from an earlier study. This suggests that Formguard can be used for large-scale web studies in different domains beyond just security and privacy.

In conclusion, Formguard combines low-level monitoring with automated interaction replay to enable continuous privacy testing on the web. Our evaluation shows that this approach is both reliable and scalable, supporting long-term operation with minimal maintenance.

Code Availability

The source code of the crawler and the recordings used for the robustness test can be found at

https://github.com/TimVlummens/Formguard

Acknowledgements

We thank Bart Preneel, Erik Poll, Mathy Vanhoef and Claudia Diaz for their useful feedback. Arlin Kokkelmans provided the list of donation pages sampled for the robustness test. Steven Wallis de Vries developed the method to handle closed Shadow DOMs in his master thesis. Tim Vlummens is funded by a research grant of the KU Leuven. This work was supported by the Flemish Government through the Cybersecurity Research Program with grant number: VOEWICS02. In addition, this work was supported by the European Commission through the Horizon 2020 research and innovation programme under grant agreement H2020-SC1-FA-DTS-2018-1-826284 ProTego, by the Research Council KU Leuven IF/C1 "From Website Fingerprinting to App Fingerprinting: Inferring private user activity from encrypted network traffic", by the Austrian Funding agency FFG under grant S3AI(COMET Module) with reference number 872172 and by the Defense Advanced Research Projects (DARPA) under contract number FA8750-19-C-0502 (RACE PRISM). Gunes Acar is supported by a Netherlands Organisation for Scientific Research (NWO) Vidi grant.

References

[1] Lawrence Abrams. Polyfill.io javascript supply chain attack impacts over 100k sites. https://www.bleepingcomputer.com/news/security/polyfillio-javascript-supply-chain-attack-impacts-over-100k-sites/, June 2024. (Online; accessed 12. Feb. 2025).

[2] Gunes Acar, Steven Englehardt, and Arvind Narayanan. No boundaries: data exfiltration by third parties embedded on web pages. In *Proceedings on Privacy Enhancing Technologies (PETS), 2020(4)*, page 220–238, 2020.

[3] Element: attachshadow() method. https://developer.mozilla.org/en-US/docs/Web/API/Element/attachShadow. (Online; accessed 28. Feb. 2025).

[4] BBC. British airways breach: How did hackers get in? https://www.bbc.com/news/technology-45446529, 07 Sep 2018. (Online; accessed 28. Feb. 2025).

[5] Ahmed Bouhoula, Karel Kubicek, Amit Zac, Carlos Cotrini, and David Basin. Automated Large-Scale analysis of cookie notice compliance. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 1723–1739, Philadelphia, PA, August 2024. USENIX Association.

[6] Chrome devtools protocol. https://chromedevtools.github.io/devtools-protocol/. (Online; accessed 28. Feb. 2025).

[7] MDN contributors. Sending form data - JavaScript —MDN. https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Forms/Sending_and_retrieving_form_data#the_action_attribute. [Online; accessed 28. Feb. 2025].

[8] MDN contributors. Function() constructor - JavaScript —MDN. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/Function, February 2025. [Online; accessed 28. Feb. 2025].

[9] crux-top-lists. https://github.com/zakird/crux-top-lists. (Online; accessed 09. Oct. 2024).

[10] Steven Wallis de Vries. leak-detect: Automatic login form leakage detection for website administrators and researchers. In *Radboud University Master Thesis*, January 2023.

[11] Debugger domain. https://chromedevtools.github.io/devtools-protocol/tot/Debugger/#event-scriptParsed. (Online; accessed 28. Feb. 2025).

[12] Debugging tests. https://playwright.dev/docs/debug#playwright-inspector. (Online; accessed 28. Feb. 2025).

[13] Kostas Drakonakis, Sotiris Ioannidis, and Jason Polakis. The cookie hunter: Automated black-box auditing for web authentication and authorization flaws. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1953–1970, New York, NY, USA, 2020. Association for Computing Machinery.

[14] email_detector.js - private relay. https://github.com/mozilla/fx-private-relay/blob/v1.2.2/extension/js/email_detector.js. (Online; accessed 05. Feb. 2025).

[15] Steven Englehardt, Jeffrey Han, and Arvind Narayanan. I never signed up for this! privacy implications of email tracking. In *Proceedings on Privacy Enhancing Technologies (PETS), 2018(1)*, page 109–126, 2018.

[16] Fathom documentation. https://mozilla.github.io/fathom/. (Online; accessed 05. Feb. 2025).

[17] General data protection regulation (gdpr). https://gdpr-info.eu/. (Online; accessed 04. Feb. 2025).

[18] Nikhil Jha, Martino Trevisan, Luca Vassio, and Marco Mellia. The internet with privacy policies: Measuring the web upon consent. In *ACM Trans. Web*, volume 16, 3, New York, NY, USA, September 2022. Association for Computing Machinery.

[19] Jaeyeon Jung, Anmol Sheth, Ben Greenstein, David Wetherall, Gabriel Maganis, and Tadayoshi Kohno. Privacy oracle: a system for finding application leaks with black box differential testing. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, page 279–288, New York, NY, USA, 2008. Association for Computing Machinery.

[20] Arlin Kokkelmans. Catching skimmer scripts in action using a hybrid analysis of javascript code. In *Radboud University Master Thesis*, September 2024.

[21] nikhiljha95, Martino Trevisan, and Antonino Musmeci. Priv-accept, 2020. https://github.com/marty90/priv-accept. (Online; accessed 04. Feb. 2025).

[22] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. You are what you include: large-scale evaluation of remote javascript inclusions. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, page 736–747, New York, NY, USA, 2012. Association for Computing Machinery.

[23] Playwright. Locators: Locate in shadow dom. https://playwright.dev/docs/locators#locate-in-shadow-dom. (Online; accessed 28. Feb. 2025).

[24] Playwright. https://playwright.dev/python/. (Online; accessed 05. Feb. 2025).

[25] Playwright - test generator. https://playwright.dev/docs/codegen. (Online; accessed 05. Feb. 2025).

[26] Asuman Senol, Gunes Acar, Mathias Humbert, and Frederik Zuiderveen Borgesius. Leaky forms: A study of email and password exfiltration before form submission. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1813–1830, Boston, MA, August 2022. USENIX Association.

[27] Asuman Senol, Alisha Ukani, Dylan Cutler, and Igor Bilogrevic. The double edged sword: Identifying authentication pages and their fingerprinting behavior. In *Proceedings of the ACM Web Conference 2024*, WWW '24, page 1690–1701, New York, NY, USA, 2024. Association for Computing Machinery.

[28] Oleksii Starov, Phillipa Gill, and Nick Nikiforakis. Are you sure you want to contact us? quantifying the leakage of pii via website contact forms. In *Proceedings on Privacy Enhancing Technologies (PETS), 2016(1)*, pages 20–33, 2016.

[29] Stripe. https://stripe.com/. (Online; accessed 06. Feb. 2025).

[30] Steven Van Acker, Daniel Hausknecht, and Andrei Sabelfeld. Measuring login webpage security. In *Proceedings of the Symposium on Applied Computing*, SAC '17, page 1753–1760, New York, NY, USA, 2017. Association for Computing Machinery.

[31] Vera Wesselkamp, Imane Fouad, Cristiana Santos, Yanis Boussad, Nataliia Bielova, and Arnaud Legout. In-depth technical and legal analysis of tracking on health related websites with ernie extension. In *Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society*, WPES '21, page 151–166, New York, NY, USA, 2021. Association for Computing Machinery.