Even Turing Should Sometimes Not Be Able To Tell: Mimicking Humanoid Usage Behavior for Exploratory Studies of Online Services

Stephan Wiefling[1], Nils Gruschka[2], and Luigi Lo Iacono[1]

[1] TH Köln - University of Applied Sciences, Cologne, Germany
{stephan.wiefling, luigi.lo_iacono}@th-koeln.de
[2] University of Oslo, Oslo, Norway
nilsgrus@ifi.uio.no

Abstract. Online services such as social networks, online shops, and search engines deliver different content to users depending on their location, browsing history, or client device. Since these services have a major influence on opinion forming, understanding their behavior from a social science perspective is of greatest importance. In addition, technical aspects of services such as security or privacy are becoming more and more relevant for users, providers, and researchers. Due to the lack of essential data sets, automatic black box testing of online services is currently the only way for researchers to investigate these services in a methodical and reproducible manner. However, automatic black box testing of online services is difficult since many of them try to detect and block automated requests to prevent bots from accessing them.
In this paper, we introduce a testing tool that allows researchers to create and automatically run experiments for exploratory studies of online services. The testing tool performs programmed user interactions in such a manner that it can hardly be distinguished from a human user. To evaluate our tool, we conducted—among other things—a large-scale research study on Risk-based Authentication (RBA), which required human-like behavior from the client. We were able to circumvent the bot detection of the investigated online services with the experiments. As this demonstrates the potential of the presented testing tool, it remains to the responsibility of its users to balance the conflicting interests between researchers and service providers as well as to check whether their research programs remain undetected.

Keywords: Black box testing, Evaluation, Testing framework

1 Introduction

The advancing digital transformation impacts all areas of human life. As a consequence, research aiming at understanding the inner workings of digital technologies, platforms, applications, services, and products, as well as their influence on human society and culture, becomes increasingly important.

Postprint version of a paper accepted for NordSec 2019. The final authenticated version is available online at https://doi.org/10.1007/978-3-030-35055-0_12.

Numerous examples for such intersections of online services with society can be found today. All sorts of social networks use non-transparent algorithms to perform content filtering and provisioning tasks, depending on one's individual characteristics and interests. Other examples can be found in the various deployed recommendation systems of e-commerce and content distribution platforms. To what extent these types of online services influence society is an important research question. Is your taste in music governed by music streaming companies and their algorithms to promote or recommend music? Are these systems exploitable for purposes other than the intended ones? First research attempts indicate that such influences on society are taking place [32,6,43].

Besides the impact on culture and society, technical aspects are more and more hidden behind the user interfaces of online services. Deployed security, as well as privacy preserving and undermining technologies, remain opaque to the user. For instance, contemporary security approaches to strengthen password-based authentication with Risk-based Authentication (RBA) [18] are deployed by only a few large online services [28,23,2], even though this technology is of broad relevance as the recent recommendation by NIST emphasizes [22]. Studying RBA-instrumented services would help to demystify RBA setups so that they can be discussed and further developed by a wider audience. This may contribute to accelerate the adoption and deployment of RBA in the wild. Good examples for exploratory research that are beneficial for society are the various studies on misusing cookies for tracking purposes [12,17,7,26,9].

An essential prerequisite to perform effective and reliable research, in this context, is the availability of data. Although many openly accessible data sets of various online services and platforms exist [4], they only provide a very limited and fragmented view. One major reason for this lack of data is that the companies and organizations possessing it—most commonly—do not share it (publicly). Thus, the digital utilities surrounding our daily lives are black boxes that do not reveal their internal workings. As this lack of transparency hinders scientific research, methods are required to methodically reverse-engineer these black boxes. This is important to understand the algorithms influencing our current and future zeitgeist as well as their corresponding security and privacy features.

Unfortunately, the investigation of the inner workings of online services is complicated for several reasons which turns studying them into a difficult problem. There is no unique path to conduct such an analysis, no simple agreed Application Programming Interface (API) or even approach. Moreover, online services are distributed systems, making the service-side inaccessible to entities other than the respective service provider itself. Also, investigating the inner workings of online services is further complicated by means of the service provider. For large-scale methodological studies, automated browsing through online services is required. However, online services integrate technical countermeasures against such automated browsing. These range from presenting CAPTCHA challenges [11] or delivering different website contents for human users and bots [1], to completely blocking the service access [30]. Hence, in order to be able to

conduct exploratory studies of online services, technologies are required to camouflage automated black box testing as far as possible.

This arms race between service providers and researchers lies in their contradicting requirements. Service providers want to keep their internals secret, as they might also contain intellectual property. Researchers instead, are keen to analyze and understand systems thoroughly, with the aim to gain knowledge and enhance system properties towards an optimum. Thus, in the absence of other means, researchers will use black box tests to determine their research results, while service providers will detect and block automated black box tests to keep their internals opaque to outsiders.

Another reason for service providers to block automated black box testing is that it is considered a double-edged sword since it might not only be used by researchers. In the hands of attackers, such testing tools can be used to threaten systems and networks, even if they are aimed at improving security. Still, as security is about balancing several trade-offs, these trade-offs need to be understood thoroughly in order to make the right compromise.

Contributions. We introduce an inspection tool to perform automated black box testing of online services and, at the same time, mimic human-like user behavior[3]. The aim is to provide a research vehicle to investigate the inner workings of online services lacking publicly accessible resources. This can foster discussion and collaboration among security researchers and service providers.

Outline. The rest of this paper is structured as follows. We review related work in Section 2. We describe the introduced inspection tool in Section 3. We give more detailed descriptions on its implementation as well as customization to study online services in Section 4. To further illustrate the use of the introduced inspection tool, Section 5 discusses exemplary studies. We discuss the benefits as well as limitations of the introduced inspection tool in Section 6. As the usage of our tool can easily be extended to exploit online services, Section 7 discusses ethical considerations before the paper concludes in Section 8.

2 Related Work

A number of researchers performed black box testing of online services with web browser automation. Choudhary et al. [10] developed a tool for automated web application testing to detect cross-browser inconsistencies on websites. Starov and Nikiforakis [34] analyzed the effect of browser extensions on the rendered Document Object Model (DOM) document for the 50 most popular websites. They showed that differences inside the DOM tree can be (mis-)used for fingerprinting and tracking the client. Englehardt and Narayanan [15] measured and analyzed one million websites and their corresponding usage of online tracking as well as the effect of browser privacy tools. Golla and Dürmuth [19] used

[3] Provided as open source software at `https://github.com/das-th-koeln/HOSIT`

browser automation to test password strength meters of online services. Degeling et al. [12] automatically extracted cookie consent notices and privacy policies of 6,579 websites inside the European Union (EU) to analyze their appearance before and after the EU General Data Protection Regulation (GDPR) [16] went into effect.

However, in all publications mentioned above, the corresponding browser automation frameworks did not aim to imitate human-like behavior as we did in our framework. As a consequence, these studies cannot tell whether their observations reflect the services' inner workings or a customized behavior due to being detected as a bot.

Other browser automation frameworks tried to imitate human-like user-actions to a small extent. Petsas et al. [29] used browser automation to evaluate the quantity of Google users with enabled Two-factor Authentication (2FA). Their framework introduced a random waiting time between clicks. Snickars and Mähler [32] analyzed the behavior of the online music streaming service Spotify with browser automation. Their automation framework conducted several user actions, e.g., logging in, selecting a track, and skipping a track. However, they noted that this was only possible before Spotify introduced reCAPTCHAs as a bot protection mechanism in 2016.

In contrast to all these frameworks, we included a considerably higher amount of efforts in our framework to closely mimic human-like behavior and bypass CAPTCHAs to not be detected as a bot (see Section 6).

The DASH tool [13] by the DETER project aimed to model human behavior in various situations, e.g., responding to phishing emails. In contrast to our tool (see Section 3), the application did not really conduct human-like actions on online services and only simulated possible behavior in theory.

Most browser automation tools described in this section were based on the Selenium framework [15,17,12,32,19,10,34]. One tool was based on CasparJS [29]. We decided to use the high-level application programming library Puppeteer [21] as a base for our tool. We chose Puppeteer over Selenium since it offers a higher-level API and is targeted to the popular Chrome browser [44] instead of multiple browsers. Note that Puppeteer was not available at the time where most of the above mentioned studies were conducted[4].

3 Humanoid Online Services Inspection Tool

The Humanoid Online Services Inspection Tool (HOSIT) was designed to simulate human-like browsing behavior on online services. While some frameworks for automated browsing are freely available on the Internet, their standard functionality makes them difficult to use for inspecting online services for several reasons:

[4] First version of the source code was published on the Puppeteer GitHub repository on May 11th, 2017: `https://github.com/GoogleChrome/puppeteer/commit/2cda8c18d10865d79d3e63b23e36aa7562098bf7`

– There is no function to create virtual identities which are perceived as real humans by online services.
– Some of the integrated functions do not model real-world human behavior and thus can be detected by online services, e.g., typing with 0 ms delay or clicking in the exact center of an element.
– The API allows activities which are not possible for real web browser users, e.g., conducting browsing activities inside two browser tabs at the same time.
– Browser automation using these frameworks can be detected due to differences between the normal and the automated browser mode [40].
– These frameworks do not log conducted actions such as name and screenshot of clicked elements automatically. This makes potential implementation errors (e.g., element with certain ID not found) hard to detect. Consequently, scaling the automation to multiple machines is difficult.

We addressed these issues with HOSIT and enhanced the integrated standard functionalities of Puppeteer with human-like browsing behavior and camouflage measures to be as indistinguishable from human users as possible:

(i) A scrolling function to imitate reading of website contents (usage can be seen in the script in Figure 2). The function scrolls down around half the display height, pauses for some time, scrolls further, pauses again and repeats this procedure until reaching the bottom of the page. We developed this function since scrolling is considered a typical behavior for human users on websites [42,36].
(ii) A function which allows switching between browser tabs. This is also a typical behavior for a human using a web browser.
(iii) A search query generator based on current events in media. The generated queries can be used to create arbitrary browsing behavior, e.g., entering query in a search engine and opening one of the results. The generator accesses a publicly available Really Simple Syndication (RSS) feed and parses the feed's content to an evaluation function which generates a list of search queries. From this list, the generator selects a random entry every time the generator is called. The search query generators can be customized and added in the HOSIT configuration, e.g., for generating search queries focused on other topics. We chose this functionality since visiting search engines is a common online activity [24,27].
(iv) Hidden element checks: some online services integrate hidden elements which can be used to detect bots, e.g., typing text inside a hidden field or clicking a hidden link. For this reason, we provide a function to check whether a certain element on a website is visible or not.
(v) Integration of external services providing CAPTCHA solving capabilities.
(vi) Automated logging of all activities conducted on the online service with screenshots into a MongoDB database for replicable studies. The database type can be adjusted for individual use case scenarios.

In general, we focused on human-like behavior that could by analyzed by reading out information via the web browser, i.e., keyboard and mouse events.

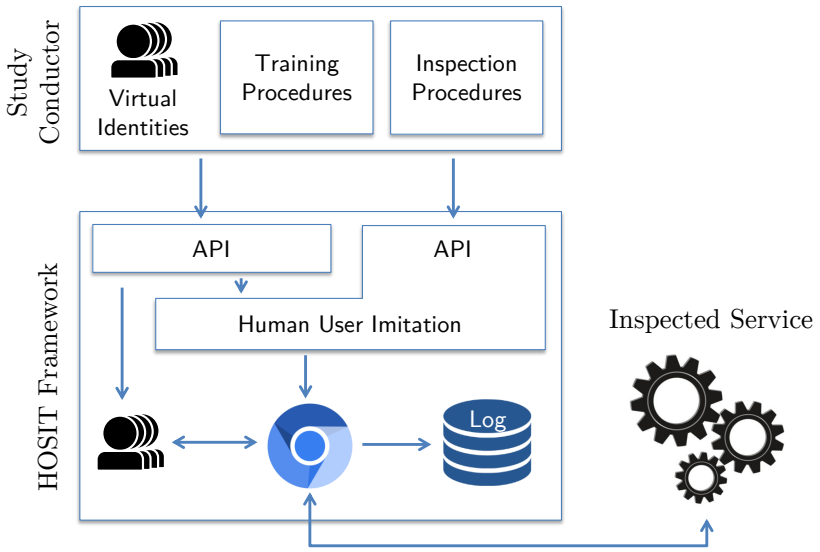

Fig. 1. Architecture of HOSIT

We considered the human-like behavior based on empirical studies modeling human computer interaction [8,14,46,37,33,25] as well as similar ideas on human behavior simulation [5].

The basic architecture of HOSIT is as follows (see Figure 1). In order to test services, the study conductor creates one or multiple virtual identities with different browsing behavior (e.g., typing speed and clicking behavior). The conductor also defines a sequence of activities to be executed on the tested services for the respective study. Examples for activities can be: "click on *shopping cart* link", "search for a friend", or "logout from service". In many cases, these activities can be divided into training procedures (let the service learn "normal" user behavior) and inspection procedures (analyze the service's reaction to unusual behavior). The HOSIT API offers functions to create virtual identities inside the HOSIT framework as well as to execute the activities. In contrast to other solutions, HOSIT enables human-like behavior in two ways. First, human-imitating behavior is automatically added to activity calls for many functions (e.g., the "click on button" function clicks on an arbitrary position inside the button). And second, HOSIT offers additional function calls allowing explicit human behavior as a part of the activity sequence (e.g., "scroll to the end of the web page"). Using a script containing a sequence of activities and the browsing behavior from the virtual identities, the HOSIT framework calls the service using a Chromium browser instance. Finally, all responses from the service are logged for later analysis.

Figure 2 shows a simple example of a HOSIT script calling an online service in a human-imitating manner. The example code invokes HOSIT to open a search

```
// Open new page tab
await controller.newPage("https://www.startpage.com/");

// Wait a random time period
await controller.randomWait();

// Click on the "Images"-Link
await controller.click("a[href='https://www.startpage.com/
    en/pics.html']");

// Wait until the text field is loaded
await controller.waitForSelector("input[type='text']");

// Generate and enter search query based on
// current events in media
await controller.typeSearchQuery("input[type='text']");

// Scroll to the bottom of the page
await controller.scrollToBottom();
```

Fig. 2. Example HOSIT script

engine, click on the link to open the image search (after waiting a random period), and enter a search query chosen randomly by HOSIT based on current events in the media. Finally, HOSIT scrolls to the bottom of the page. This results in a usage of the online service in a way that a human would also do.

All activities performed on the online service as well as errors are logged into a database. As a result, study conductors get an overview of all interactions that the identities performed on the online service. This also eases the debugging of errors caused by activities of a certain identity.

4 Implementation

We implemented HOSIT using the Node.js library Puppeteer [21] in version 0.13.0 for browser automation. Consequently, HOSIT can be used on all operating systems which are capable of running the Node.js runtime environment and the browser Chromium. A Chromium version is bundled with the HOSIT installation. Nevertheless, HOSIT can be configured to use a customized Chromium or Chrome browser version instead of the bundled version. This might be necessary for example, when testing websites requiring Digital Rights Management (DRM) functionalities, which are included in Chrome but not in Chromium.

Chromium is executed in a custom *headful* mode, in which the browser is launched in the standard mode with visible GUI[5]. HOSIT uses this headful

[5] To be compatible with Linux servers or Docker containers without a visible desktop environment, the headful mode can also be run inside a virtual window session.

mode to minimize the detection of automated browsing. Chromium's headless mode, which is designed specifically for browser automation, can be detected by a number of differences in the browser's properties and behavior [40]. During testing we actually experienced that online services treat headless browsers differently. Amazon, for example, required a CAPTCHA in headless but not in headful mode. We also patched HOSIT against known headless browser detection mechanisms [40]. HOSIT executes these patches when launching Chromium, e.g., removing the `navigator.webdriver` property [35].

During testing, we found some indications that browser automation can be detected with the standard functionality of Puppeteer. For instance, Amazon rated correctly entered CAPTCHA solutions as *not correct* if "typed" in by the standard Puppeteer function. Therefore, we enhanced some of Puppeteer's integrated functions with human-like user behavior. We compared manual browsing behavior with the automated behavior of Puppeteer to determine differences and optimized the affected functions. First, we modified the constant standard delays between pressing and releasing key buttons with randomized delays. These delays vary with an average typing speed which is defined by the identity (average time and maximum deviation). We recommend to measure these delays on real humans before setting them on the identities. By default, we set empirically measured typing speeds [8,14] on the identities. This procedure helped mimic human behavior more precisely. Further, we modified the mouse input behavior. Instead of clicking in the exact center of an element, the mouse selected a random click point in the center quarter of the element. We also replaced the default delay between pressing and releasing the left mouse button of 0 ms with an empirically measured clicking time with randomized variations [25].

We, moreover, added further functionalities to HOSIT which did not exist in Puppeteer (see Table 1). Finally, we simplified the API of Puppeteer and added recurrent tasks for the use case scenario inside the functions, e.g., automatically adjust the browser resolution when creating a new tab. As a result, fewer function calls are required to achieve the same result as with Puppeteer while being more human-like in many respects.

As stated in Section 3, each HOSIT instance is linked to a virtual identity which controls a browser instance. All further browsing behaviors are derived from this identity on this instance (e.g., typing behavior, selecting different categories based on the virtual identity's persona). The identity manages all browser tabs such as opening, switching, and closing browser tabs, and performs the actions on the website. These actions range from typing or clicking to scrolling and can only be performed in the currently open browser tab. We decided to select this identity-based structure to both optimize the API for the use case and to avoid unrealistic browsing behavior that was possible in Puppeteer, e.g., clicking buttons in two browser tabs at the same time.

When developing own studies of online services, study conductors have to design individual testing procedures with HOSIT. This is necessary since navigation structures and functionalities differ between online services and might change over time. For fine-grained variations of the browsing behavior, each

Table 1. Feature differences between Puppeteer and HOSIT

	Puppeteer 0.13.0	HOSIT
Properties		
Typing speed	Constant	Randomized variations
Click position	Exact element center	Randomized variations
Click time	0 ms	Realistic [25]
Logging	Limited	Extended*
Browsing behavior changes	-	Yes, based on persona
Bot detection protection	-	Patched
Functions		
Common workflows	Need to be repeated	Integrated in Controller class
Search query generator	-	Included
CAPTCHA solving	-	Included
Scrolling	-	Included
Select tabs	-	Included

- Not included
* Logs all conducted actions with screenshots into a database

HOSIT instance provides functions which can be used to increase randomized browsing behavior. These functions range from providing a random boolean value with a given probability for if-else conditions to providing the persona of the identity. By using these functions, we achieved that each browser session performed by HOSIT appeared differently on the tested online services.

5 Exemplary Use

To evaluate HOSIT, we conducted two studies that we discuss in the following. Both experiments would not have been possible without HOSIT or just with significant higher effort. The discussions will also provide a better understanding of HOSIT deployments based on the two given exemplary use case scenarios.

5.1 Use Case 1: RBA

Risk-based Authentication (RBA) [18] is an adaptive security measure to improve password authentication. During login, RBA monitors and stores additional features available in the context (e.g., IP address or user agent string) and requests additional information for authentication if a certain risk level is exceeded. RBA offers protection against security risks such as credential stuffing, password database leaks and intelligent password guessing methods. Beyond

that, RBA has the potential to compensate low adoption rates of Two-factor Authentication (2FA). For instance, less than 10% of all active Google users activated 2FA in January 2018 [28].

RBA is recommended in the NIST digital identity guidelines [22] and is used by several large-scale online services. However, these online services keep their implementations secret and restrain their approaches for a public discussion in science. This lack of public knowledge makes it difficult for small and medium websites to use RBA.

For this reason, we black box tested eight popular online services[6] with HOSIT to find out more about the corresponding RBA implementations, i.e., features and offered additional authentication factors [45]. We created 28 virtual online identities, registered 224 user accounts with the eight targeted services, and observed the services' behavior when accessing them under different circumstances. Each virtual identity had its own unique IP address from the same Internet service provider and a personal computer.

However, analyzing the inner workings of RBA is complicated, since one of the main tasks of RBA is to protect against bots. During pilot testing, we found indicators that some online services treated an automated browser using Puppeteer differently. For this reason, we designed our study using HOSIT to imitate human user behavior as exact as possible. Imitating human behavior was essential to make sure that the observed services' behavior is identical to normal usage.

RBA estimates the login risk based on the login history of the user. Therefore, our virtual identities conducted 20 browsing sessions including user sessions on the online services. The user sessions included login, activities on the online service, and logout. After these 20 browsing sessions, we varied browser features including the login time, IP address and device, logged in again on all online services, and observed the reactions. Based on the reactions, we drew conclusions about the inner RBA workings of the tested online services. The activities on the online services were randomized and individualized with HOSIT and differed on each of the online services. We selected typical activities for each of the online services, e.g., scrolling in the newsfeed, checking mail inbox or browsing for articles or jobs. In addition, these activities included a lot of randomness to mitigate being detected as a bot. As an example, on social media websites, it was randomly alternated between scrolling in the newsfeed, checking the message inbox and searching for content.

Since online services are likely tracking their users [7,9], all virtual identities simulated randomized browsing behavior in each browsing session with HOSIT. They visited search engines and entered search queries based on current topics discussed in media. Then, they opened some of the websites and "read" the text by scrolling and waiting. Also, the testing sequence of services was shuffled to a random order. This was done to prevent our virtual online identities from logging into the online services at similar times.

[6] Amazon, Facebook, GOG.com, Google, iCloud, LinkedIn, Steam and Twitch

With the study based on HOSIT, we were able to derive features as well as an approximation to the respective weightings used for the RBA risk estimation of popular online services. One major finding was that five of the eight tested popular online services used RBA. Also, each of the services had a different RBA implementation, varying from protecting all users to only a selection of users. Besides using the IP address as a high weighted RBA feature, some services also used additional lower weighted features (e.g., user agent string).

More details on the RBA study can be found in the original publication [45].

5.2 Use Case 2: Amazon Product Recommendation System

When shopping on Amazon, a large amount of customer actions are tracked by the online shop. Besides the purchased items, these actions also include every item just visited by the user. Details can be seen in logs which European users can request from Amazon [3]. This right to request all personal data stored on a service provider is granted by the GDPR [16].

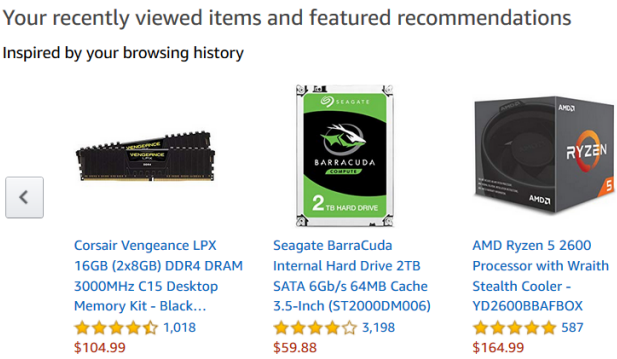

Fig. 3. Shopping history and recommendations in the Amazon online shop

Amazon offers different types of product recommendations that are considered interesting for the customer [31]. When visiting a product page for example, similar or related items are presented. These items are based on sponsoring (*"Sponsored products related to this item"*) or shopping behavior of other users (*"Customers who bought this item also bought"*). Another recommendation type (*"Inspired by your browsing history"*) is based on the user's own browsing history mentioned in the previous paragraph, i.e., not only the purchase history, but also items just visited.

The recommendations given by Amazon are interesting for customers as well as other online shops. Hence, these recommendations can be considered a valuable asset for Amazon. It is therefore a reasonable assumption that Amazon, by detecting bots, is protecting these assets from automatic scraping. As a countermeasure, a bot could be presented different website content compared to human

users, e.g., a CAPTCHA, different recommendations, or even recommendations with different prices. Thus, research on recommendations shown to human users requires a human-imitating client as provided by HOSIT.

In order to analyze the recommendation system and to verify this assumption, we conducted a study on the Amazon online shop. In this study, our (automated) user requested a fixed sequence of products and recorded the recommended products on the history page.

We conducted the same study with three different types of clients: automatically using Puppeteer, automatically using HOSIT, and manually by a human user. In addition, the products were requested in two different manners: either by simply opening the sequence of product page URLs or with "human like" online shopping, i.e., typing a search term into the search bar, selecting a search result, looking at the product page, searching for a next item and so forth. Finally, we performed this study with both registered and unregistered Amazon users.

The evaluation of the study revealed an unexpected result: the recommended items were exactly the same in all cases, including the order of items and the product prices. Thus, in contrast to the RBA of Amazon services, we assume that Amazon does not perform any bot detection for their recommendation system or allows bots to a certain degree, e.g., let harmless bots pass, block bots exaggerating the network traffic [1].

6 Benefits and Limitations

We put a lot of effort into ensuring that our tool was not recognized as a bot by online services. Nevertheless, the possibility that online services recognize HOSIT-based experiments as automated browsing remains. Even human-like browsing if performed constantly for a very long time will surely be detected. Also, creating too many new user accounts from the same IP address in a short time is likely to be noticed and even stopped by many online services. This, however, is even true when performed by a human. Thus, despite all protection mechanisms, automated browsing activities should not be exaggerated and kept at a realistic level, e.g., by introducing a long pause after some hours.

Still, based on our observations, we are convinced that our tool remained under respective bot detection thresholds. For instance, Amazon did not block automated logins with HOSIT while it did with Puppeteer. In March 2019, we also tested HOSIT using an instance of reCAPTCHA v3 [20], which is specifically designed to recognize bots. It analyzed the browsing behavior and returned a risk score. The score was a numerical value between 1.0 (*very likely a human*) and 0.0 (*very likely a bot*). We opened a testing website, which used reCAPTCHA v3, with both Puppeteer and HOSIT, and observed the risk score returned by the reCAPTCHA API. When using HOSIT, the reCAPTCHA v3 risk scores were identical to those of a human-controlled Chrome browser with empty browsing history and cookies (score: 0.7 = *likely a human*), while this was not the case with Puppeteer (score: 0.1 = *likely a bot*). After the release of HOSIT in April 2019, the reCAPTCHA risk score when using HOSIT was lowered to 0.3. This

again underlines the arms race between bot detectors and bot detection avoiders. We will observe novel bot detection mechanisms and integrate countermeasures against them in future versions of HOSIT.

Before conducting research studies with HOSIT, study conductors are advised to test for anomalies on online services. In addition, study conductors should monitor which JavaScript attributes were read by online services while accessing this service [41]. These tests are helpful to determine possible bot detection and to implement countermeasures as a result.

Overall, we still find HOSIT highly sensible for studies due to the following reasons: (i) The reCAPTCHA v3 risk score is still higher than with Puppeteer. (ii) Not all online services on the Internet use the current reCAPTCHA. (iii) The API of HOSIT is more simplified than the API of comparable tools such as Selenium and Puppeteer, which makes it much easier to use.

7 Ethical Considerations

As with most tools for black box analysis, HOSIT is considered as "dual use", i.e., it can be used for illegitimate purposes as well. On the one hand, it can be beneficial to gather information on service behavior determining our everyday life. On the other hand, it could also be used for click fraud on online advertising, theft of intellectual property, or possibly even denial of service. Further, when using HOSIT, researchers should carefully check not to violate the respective *Terms of Service*. Also, researchers should use automated browsing responsibly, e.g., by keeping the impact on the inspected online services minimal [29,45].

We believe, however, that the results gathered by public research with HOSIT can be beneficial for a large user base and thus should be set ahead of corporate goals. We further argue that our work is justified, as the expected gain from scientific studies outweighs the potential security implications. Ultimately, we hope that public research based on our inspection tool will be beneficial for smaller online services. In consequence, security related research using this tool will protect a larger user base.

8 Conclusion

In this paper we presented HOSIT, a framework for automatically invoking online services in a human-like manner. As many online services try to detect if the client is a person or a bot, human-imitating behavior is required for automated service interactions in order to receive the same results as a human user. HOSIT implements a number of human-like behavior techniques and can be extended with further methods, as required by the targeted experiment and online service.

HOSIT can be used to circumvent services' bot-detection and to perform large-scale research on how online services behave towards human users. This is particularly interesting if the offered service depends—or is suspected to depend—on the user's behavior, location, history, device, and so on. Examples for such services are results from search engines, information in social networks,

or recommendations in online shops. In particular, our research on RBA [45], which led to valuable and beneficial results, would not have even been possible without HOSIT. We discovered—among others—a privacy leakage in one of the RBA dialogs of Facebook and resolved this issue within a responsible disclosure process.

In future work, we will continuously extend and refine the human-imitating techniques of HOSIT. To evaluate their effectiveness, we will perform in-depth analysis on the influence of our methods on bot-detection systems, such as re-CAPTCHA, on a regular basis. Moreover, we will apply HOSIT to study further scenarios including, e.g., search engine results and local browser storage usage patterns. We hope to see more of such research conducted on the basis of HOSIT.

For future research on service behavior, we will also follow alternative approaches. Instead of performing black box tests using camouflaged tools, services could enable responsible access to researchers. Researchers would benefit from unbiased results and focus on the analysis (and not on the black box testing tools), and services could advertise their support for research. This responsible service access could be monitored by an independent organization or public authority. A similar method called *regulatory sandbox* is used successfully in the financial area [38] and is currently discussed for research on personal identifying information [39].

Acknowledgements. We would like to thank Tanvi Patil for proofreading a draft of the paper. This research was supported by the research training group "Human Centered Systems Security" (NERD.NRW) sponsored by the state of North Rhine-Westphalia.

References

1. Akamai: Bot-Manager (Jan 2018), `https://www.akamai.com/us/en/multimedia/documents/product-brief/bot-manager-product-brief.pdf`
2. Allen, N.A.: Risk based authentication, patent number US9202038B1 (2015)
3. Amazon: Amazon.co.uk Help: How do I request my data? (2019), `https://www.amazon.co.uk/gp/help/customer/display.html?nodeId=G5NBVNN2RHXD5BUW`
4. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: A Nucleus for a Web of Open Data. In: The Semantic Web, vol. 4825, pp. 722–735. Springer Berlin Heidelberg (Nov 2007)
5. Blythe, J., Botello, A., Sutton, J., Mazzaco, D., Lin, J., Spraragen, M., Zyda, M.: Testing Cyber Security with Simulated Humans. In: IAAI '11. San Francisco, CA, USA (Aug 2011)
6. Bond, R.M., Fariss, C.J., Jones, J.J., Kramer, A.D.I., Marlow, C., Settle, J.E., Fowler, J.H.: A 61-million-person experiment in social influence and political mobilization. Nature 489(7415), 295–298 (Sep 2012)
7. Bujlow, T., Carela-Espanol, V., Lee, B.R., Barlet-Ros, P.: A survey on web tracking: Mechanisms, implications, and defenses. Proceedings of the IEEE 105(8), 1476–1510 (Aug 2017)

8. Card, S.K., Moran, T.P., Newell, A.: The keystroke-level model for user performance time with interactive systems. Communications of the ACM 23(7), 396–410 (Jul 1980)
9. Chaabane, A., Kaafar, M.A., Boreli, R.: Big friend is watching you: Analyzing online social networks tracking capabilities. In: WOSN '12. pp. 7–12. ACM, Helsinki, Finland (Aug 2012)
10. Choudhary, S.R., Prasad, M.R., Alessandro Orso: X-PERT: a web application testing tool for cross-browser inconsistency detection. In: ISSTA '14. pp. 417–420. ACM, San Jose, CA, USA (2014)
11. Dalai, A.K., Jena, S.K.: Online identification of illegitimate web server requests. In: ICIP '11. pp. 123–131. Springer, Bangalore, India (2011)
12. Degeling, M., Utz, C., Lentzsch, C., Hosseini, H., Schaub, F., Holz, T.: We Value Your Privacy ... Now Take Some Cookies: Measuring the GDPR's Impact on Web Privacy. In: NDSS '19. San Diego, CA, USA (Feb 2019)
13. DETER Project: DASH user guide (2014), https://deter-project.org/sites/deter-test.isi.edu/files/files/dash_users_guide.pdf
14. Drury, C.G., Hoffmann, E.R.: A model for movement time on data-entry keyboards. Ergonomics 35(2), 129–147 (Feb 1992)
15. Englehardt, S., Narayanan, A.: Online Tracking: A 1-million-site Measurement and Analysis. In: CCS'16. pp. 1388–1401. ACM, Vienna, Austria (Oct 2016)
16. European Parliament and Council: Regulation (EU) 2016/679 (GDPR) (Jan 2016), http://data.europa.eu/eli/reg/2016/679/oj/eng
17. Franken, G., Goethem, T.V., Joosen, W.: Who Left Open the Cookie Jar? A Comprehensive Evaluation of Third-Party Cookie Policies. In: USENIX Security '18. Baltimore, MD, USA (Aug 2018)
18. Freeman, D., Jain, S., Duermuth, M., Biggio, B., Giacinto, G.: Who Are You? A Statistical Approach to Measuring User Authenticity. In: NDSS '16. San Diego, CA, USA (Feb 2016)
19. Golla, M., Dürmuth, M.: On the Accuracy of Password Strength Meters. In: CCS '18. pp. 1567–1582. ACM, Toronto, Canada (Oct 2018)
20. Google: reCAPTCHA v3 (Jul 2019), https://developers.google.com/recaptcha/docs/v3
21. Google Chrome: Puppeteer - Headless Chrome node API (Jul 2019), https://github.com/googlechrome/puppeteer
22. Grassi, P.A., Fenton, J.L., Newton, E.M., Perlner, R.A., Regenscheid, A.R., Burr, W.E., Richer, J.P., Lefkovitz, N.B., Danker, J.M., Choong, Y.Y., Greene, K.K., Theofanos, M.F.: Digital identity guidelines: authentication and lifecycle management. Tech. Rep. NIST SP 800-63b, National Institute of Standards and Technology, Gaithersburg, MD (Jun 2017)
23. Iaroshevych, O.: Improving Second Factor Authentication Challenges to Help Protect Facebook account owners. In: SOUPS '17. USENIX Association, Santa Clara, CA, USA (Jul 2017)
24. Judd, T., Kennedy, G.: A five-year study of on-campus Internet use by undergraduate biomedical students. Computers & Education 55(4), 1564–1571 (Dec 2010)
25. Komandur, S., Johnson, P.W., Storch, R.: Relation between mouse button click duration and muscle contraction time. In: EMBC '08. IEEE (Aug 2008)
26. Li, T.C., Hang, H., Faloutsos, M., Efstathopoulos, P.: TrackAdvisor: Taking Back Browsing Privacy from Third-Party Trackers. In: Passive and Active Measurement, vol. 8995, pp. 277–289. Springer International Publishing, Cham (2015)
27. Mark, G., Wang, Y., Niiya, M.: Stress and multitasking in everyday college life: an empirical study of online activity. In: CHI '14. ACM, Toronto, Canada (2014)

28. Milka, G.: Anatomy of Account Takeover. In: Enigma 2018. USENIX Association, Santa Clara, CA (Jan 2018), https://www.usenix.org/node/208154
29. Petsas, T., Tsirantonakis, G., Athanasopoulos, E., Ioannidis, S.: Two-factor authentication: Is the world ready?: Quantifying 2FA adoption. In: EuroSec '15. pp. 4:1–4:7. ACM, Bordeaux, France (Apr 2015)
30. Rsmwe: Rakuten.com Chrome Headless Detection (Feb 2018), https://github.com/Rsmwe/Headless-detected-demo
31. Smith, B., Linden, G.: Two Decades of Recommender Systems at Amazon.com. IEEE Internet Computing 21(3), 12–18 (May 2017)
32. Snickars, P., Mähler, R.: SpotiBot — Turing Testing Spotify. Digital Humanities Quarterly 12, 12 (2018)
33. Soukoreff, R.W., MacKenzie, I.S.: Towards a standard for pointing device evaluation, perspectives on 27 years of Fitts' law research in HCI. International Journal of Human-Computer Studies 61(6), 751–789 (Dec 2004)
34. Starov, O., Nikiforakis, N.: XHOUND: Quantifying the Fingerprintability of Browser Extensions. In: IEEE S&P. IEEE, San Jose, CA, USA (May 2017)
35. Steward, S., Burns, D.: WebDriver - W3C Recommendation (Jun 2018), https://www.w3.org/TR/webdriver1/
36. Sulikowski, P., Zdziebko, T., Turzyński, D., Kańtoch, E.: Human-website interaction monitoring in recommender systems. Procedia Computer Science 126, 1587–1596 (2018)
37. Trauzettel-Klosinski, S., Dietz, K.: Standardized Assessment of Reading Performance: The New International Reading Speed Texts IReST. Investigative Opthalmology & Visual Science 53(9), 5452 (Aug 2012)
38. UK Financial Conduct Authority: Regulatory Sandbox Lessons Learned Report (2017), https://www.fca.org.uk/publication/research-and-data/regulatory-sandbox-lessons-learned-report.pdf
39. UK Information Commissioner's Office: Call for Views on Building a Sandbox: Summary of Responses and ICO Comment (2018), https://ico.org.uk/media/about-the-ico/consultations/2260322/201811-sandbox-call-for-views-analysis.pdf
40. Vastel, A.: Detecting Chrome headless, new techniques (Jan 2018), https://antoinevastel.com/bot%20detection/2018/01/17/detect-chrome-headless-v2.html
41. Vastel, A.: How to monitor the execution of JavaScript code with Puppeteer and Chrome headless (Jun 2019), https://antoinevastel.com/javascript/2019/06/10/monitor-js-execution.html
42. Velayathan, G., Yamada, S.: Behavior-Based Web Page Evaluation. In: WI-IAT '06. pp. 409–412 (Dec 2006)
43. Venkatadri, G., Lucherini, E., Sapiezynski, P., Mislove, A.: Investigating sources of PII used in Facebook's targeted advertising. PETS 2019, 227–244 (Jan 2019)
44. W3Schools: Browser Statistics: The Most Popular Browsers (2019), https://www.w3schools.com/browsers/default.asp
45. Wiefling, S., Lo Iacono, L., Dürmuth, M.: Is This Really You? An Empirical Study on Risk-Based Authentication Applied in the Wild. In: IFIP SEC '19. Springer International Publishing (Jun 2019), https://doi.org/10.1007/978-3-030-22312-0_10
46. Williams, J.L., Skinner, C.H., Floyd, R.G., Hale, A.D., Neddenriep, C., Kirk, E.P.: Words correct per minute: The variance in standardized reading scores accounted for by reading speed. Psychology in the Schools 48(2), 87–101 (Feb 2011)