PhishInPatterns: Measuring Elicited User Interactions at Scale on Phishing Websites

Karthika Subramani
ks54471@uga.edu
University of Georgia
Athens, United States

William Melicher
bmelicher@paloaltonetworks.com
Palo Alto Networks
Santa Clara, United States

Oleksii Starov
ostarov@paloaltonetworks.com
Palo Alto Networks
Santa Clara, United States

Phani Vadrevu
pvadrevu@uno.edu
University of New Orleans
New Orleans, United States

Roberto Perdisci
perdisci@uga.edu
University of Georgia and Georgia Institute of Technology
Athens, United States

ABSTRACT

Despite phishing attacks and detection systems being extensively studied, phishing is still on the rise and has recently reached an all-time high. Attacks are becoming increasingly sophisticated, leveraging new web design patterns to add perceived legitimacy and, at the same time, evade state-of-the-art detectors and web security crawlers.

In this paper, we study phishing attacks from a new angle, focusing on how modern phishing websites are designed. Specifically, we aim to better understand what type of user interactions are elicited by phishing websites and how their user experience (UX) and interface (UI) design patterns can help them accomplish two main goals: i) lend a sense of professionalism and legitimacy to the phishing website, and ii) contribute to evading phishing detectors and web security crawlers. To study phishing at scale, we built an intelligent crawler that combines browser automation with machine learning methods to simulate user interactions with phishing pages and explore their UX and UI characteristics. Using our novel methodology, we explore more than 50,000 phishing websites and make the following new observations: i) modern phishing sites often impersonate a brand (e.g., Microsoft Office), but surprisingly, without necessarily cloning or closely mimicking the design of the corresponding legitimate website; ii) they often elicit personal information using a multi-step (or multi-page) process, to mimic users' experience on legitimate sites; iii) they embed modern user verification systems (including CAPTCHAs); and ironically, iv) they sometimes conclude the phishing experience by reassuring the user that their private data was not stolen. We believe our findings can help the community gain a more in-depth understanding of how web-based phishing attacks work from a users' perspective and can be used to inform the development of more accurate and robust phishing detectors.

IMC '22, October 25–27, 2022, Nice, France

ACM ISBN 978-1-4503-9259-4/22/10…$15.00
https://doi.org/10.1145/3517745.3561467

CCS CONCEPTS

• **Security and privacy → Phishing**; *Usability in security and privacy*; • **Computing methodologies → Neural networks**.

KEYWORDS

Phishing, Neural Networks, Crawler, User Experience, Captcha

ACM Reference Format:
Karthika Subramani, William Melicher, Oleksii Starov, Phani Vadrevu, and Roberto Perdisci. 2022. PhishInPatterns: Measuring Elicited User Interactions at Scale on Phishing Websites. In *ACM Internet Measurement Conference (IMC '22), October 25–27, 2022, Nice, France.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3517745.3561467

1 INTRODUCTION

Despite the fact that phishing attacks have been studied extensively and a variety of detection systems been proposed [11–13, 15, 22, 23, 26, 36, 45], phishing is still on the rise and has recently reached an all-time high. For instance, according to a recent report [3] by Cisco, over 90% of data breaches occur due to phishing attacks. To evade defenses, attackers develop increasingly sophisticated phishing websites that leverage modern web design patterns to attain a high level of perceived legitimacy and, at the same time, evade state-of-the-art detectors [11, 26] and web security crawlers [45].

In this paper, we study phishing attacks from a new angle. Specifically, we aim to provide a more in-depth understanding of how modern web-based phishing attacks work from a users' perspective, as a way to provide insights that can inform the development of more accurate and robust defenses. To this end, we study how modern phishing websites are designed, to better understand what type of user interactions are elicited by these malicious websites and how their user experience (UX) and interface (UI) design patterns may help them concurrently accomplish two main goals: i) lend a sense of professionalism and legitimacy to the phishing website, and ii) contribute to evading phishing detectors and web security crawlers.

Liu et al. [26] have very recently proposed an initial step towards leveraging information about web UI input elements to improve phishing detection. The authors propose to identify the *intent* of phishing pages to steal users' credentials. By combining visual UI analysis to identify credentials input boxes with the detection of brand-related logos, the proposed detector is able to outperform

previous visual phishing detectors. However, the proposed system is limited to only identifying pages that request login credentials using a combination of computer vision-based analysis and heuristics (e.g., specific keywords within submission buttons). In our study, we find that many modern phishing websites (around 28%) do not present users with a simple first page that contains the page logo and login credentials request. Rather, they often require some form of "click through" user verification, before reaching a page that requests user data. Furthermore, many phishing websites do not ask the user for login credentials at all (see example in Figure 11, in Appendix). Therefore, we believe that the findings from our study can be used to significantly strengthen future defenses, including [26].

User inputs required on phishing websites have also been partially explored in recent work on web security crawlers. For instance, CrawlPhish [45] explores how modern phishing websites implement cloaking to evade analysis by security crawlers. This includes *user interactions*, such as pop-ups, mouse movement, and click-through actions. However, while CrawlPhish focuses only on cloaking, our work analyzes the user interactions elicited by phishing websites beyond the first user verification page. Furthermore, in our study we found that only about 9% of phishing sites do require some form of user verification to navigate beyond an initial page, before requesting some form of user data. By exploring complex phishing attacks at multiple stages, our study provides both new insights on user verification interactions and an in-depth analysis of victims' experience on modern phishing websites.

To study phishing websites at scale, we developed an intelligent phishing website crawler that combines browser automation with machine learning methods to infer the inputs elicited by these websites, mimic user interactions with phishing pages, and explore their UX and UI characteristics. Starting from a large commercial feed of live phishing websites consisting of more than 50,000 phishing URLs collected between March 20th, 2022 and May 1st, 2022, our crawler is programmed to interact with each phishing website, automatically identify and interpret its UI elements using machine learning methods (e.g., OCR, object detection and classification), forge and enter syntactically valid data into the identified input fields, and along the way collect information to enable an at-scale analysis of the UX flow that real users would encounter on these websites. By intelligently simulating user actions, we are able to gain deeper insights into the entire phishing attack UX from start to finish. An overview of our measurement system and data analysis process is provided in Figure 6.

Using our novel measurement methodology to explore the more than 50,000 phishing websites in our data feed, we make the following new main observations: i) modern phishing sites often impersonate a brand (e.g., Microsoft Office), but surprisingly, without necessarily cloning the design of the corresponding legitimate website; ii) they often elicit personal information using a multi-stage (or multi-page) process, to mimic users' experience on legitimate sites; iii) they embed modern user verification systems (including CAPTCHAs); and ironically, iv) they sometimes conclude the phishing experience by reassuring the user that their private data was not stolen.

In summary, we make the following contributions:

- We study phishing from a new angle to better understand what type of user interactions are elicited by phishing websites and what user experience (UX) and interface (UI) design patterns they use. The results from our study can help inform the development of more accurate and robust phishing detectors.
- To study phishing websites at scale, we developed an intelligent phishing website crawler that combines browser automation with machine learning methods to infer the inputs elicited by these websites, mimic user interactions with phishing pages, and explore their UX and UI characteristics. As phishing websites are naturally adversarial, our crawler aims to overcome a number of challenges related to how phishing website attempt to hide content from automated analysis (e.g., crawlers that use simple HTML parsing).
- Using our novel measurement system, we interact with and collect data from more than 50,000 live phishing websites. For instance, we found that 45% of phishing sites use a multi-page data stealing design pattern, mimicking user data gathering on legitimate websites. Furthermore, 9% of modern phishing sites use different forms of initial user verification before navigating the user to the data collection pages.
- We have made our system publicly available[1] to foster future research in this area.

2 MODERN PHISHING - EXAMPLES AND CHALLENGES

In this section, we discuss a few representative examples that demonstrate some of the UX and UI characteristics of modern phishing websites, and discuss the challenges that these phishing sites pose to at-scale measurements.

(a) (b)

Figure 1: (a) Real-world phishing site targeting the DHL brand; (b) Login page on legitimate DHL website.

Brand impersonation does not require cloning. Figure 1a shows an example of a recent phishing page that impersonates the DHL brand. While the design is quite convincing, it does not actually closely mimic the true DHL website (figure 1b); it is sufficient for it to be believable.

Challenges: Measuring the number of phishing sites that do not closely mimic their legitimate counterpart is difficult. The following main ingredients are needed: i) determine the brand that is impersonated by a phishing site, and ii) determine if the phishing page closely resembles the impersonated website (e.g., visually and/or in its HTML structure). We address these challenges by leveraging the

[1] https://github.com/karthikaS03/PhishInPattern

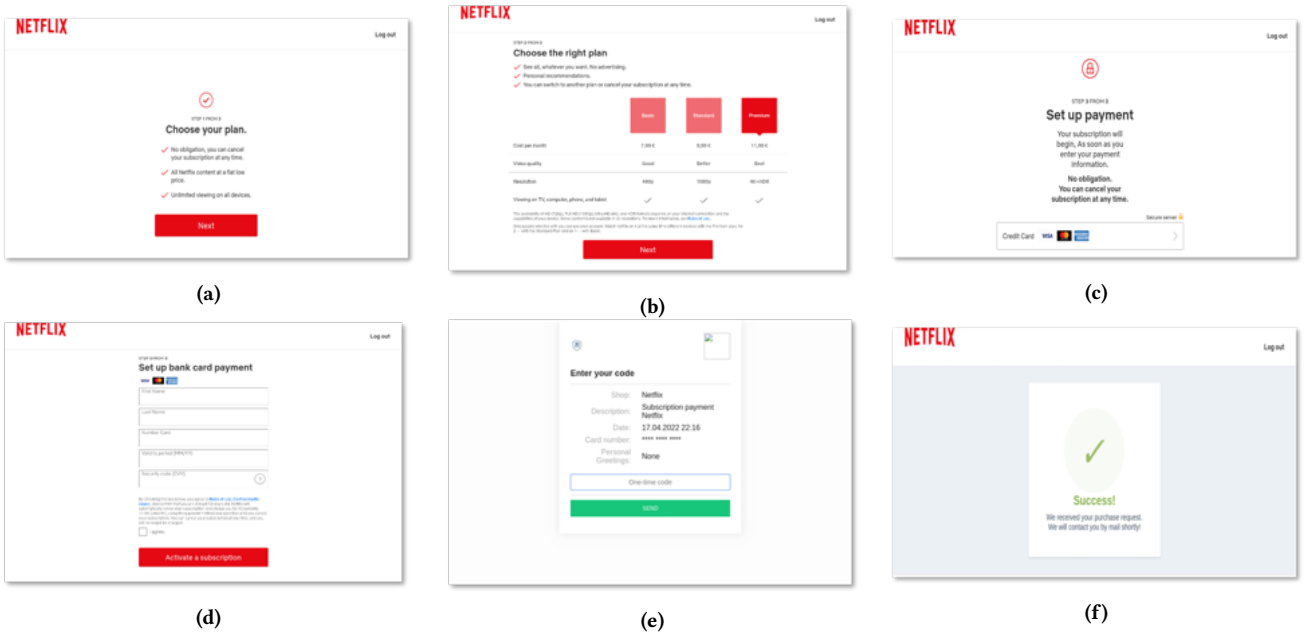


Figure 2: Example of multi-step phishing website that impersonates Netflix. (a) Page 1: *click-through* page; (b) Page 2: *click-through* page; (c) Page 3: Subscription Setup Page; (d) Page 4: Stealing financial information; (d) Page 5: OTP confirmation page; (e) Page 6 (UX termination): *Confirmation* message for submitting the (now stolen) data.

brand labeling provided by our phishing data feed, and by leveraging VisualPhishNet [11] to determine if the phishing site is similar to the legitimate brand website or not, as explained more in detail in Section 5.

Input identification and inference obstacles. One of the key characteristics of phishing websites is the fact that they focus on stealing users' data, and thus must have a way for a user to input and submit information. This is most typically accomplished by having the user submit a web form, which can include several input boxes and drop-down menus.

Challenges: Although the phishing page must include input boxes and visually impersonate a legitimate brand, there is no restriction on the way that this is accomplished. The same rendering of a web page can be achieved with different HTML and JS code. Obviously, the phishing site designers have an interest in making automated parsing of their pages as difficult as possible. For instance, Figure 3a shows a phishing page that impersonates the USAA's website. The page includes a form requesting the user's personal information. However, this page was built by using a background image containing the name of the input box fields, as shown in Figure 3b. Then, the input boxes are simply positioned on top of the background image. Clearly, this makes the automated interpretation of the page challenging, although visually the page looks perfectly believable to potential victims. To address this and similar challenges, we use a combination of HTML parsing and OCR to infer the location and type of data requested by each input box.

Multi-stage phishing. Another approach that the attackers use to mimic the UX on legitimate sites is to elicit detailed user information in a multi-step (or multi-page) process. This is to resemble what

(a) (b)

Figure 3: Example of real-world phishing site using a background image to evade form parsing. (a) Full page after rendering. (b) Background image used to render the form.

many users experience on legitimate sites, wherein the website provides the user with a brief explanation before asking for more sensitive data, such as payment information. Figure 2 shows an example of a phishing campaign that does just that. The phishing website impersonates Netflix, and includes multiple pages that let the user subscribe to a (fake) service, assure the user that they can cancel the service at anytime (at step 3), and proceed to collect payment information.

Interestingly, some phishing websites are designed with an end-to-end UX flow that includes a purposeful termination phase. For instance, after the user enters the required personal information, they might be shown a message of "congratulations" or confirmation of success, as shown in the example in Figure 2f. In other cases, the phishing site may simply redirect the user to the legitimate brand website. More notably, we also observed a number of

phishing campaigns that terminate the phishing experience with a reassuring message that their data was *not* phished (Figure 4).

Challenges: To be able to automatically navigate through the entire UX flow, we need to automatically interpret what the page is asking, enter plausible and correctly formatted data, activate the *submit* form action, and collect and analyze all details about the pages encountered along these multi-stage phishing attacks. We discuss our approach towards addressing this problem in Section 4.

Figure 4: Examples of fake phishing awareness training messages.

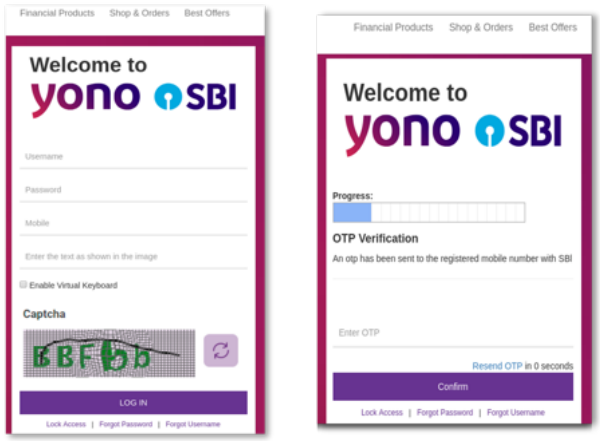
User verification. Some modern phishing websites do not present the user with a page that immediately requests for data (e.g., via a web form on the first encountered page). Rather, in some cases, the phishing site designers will present the user with a message that the user needs to read and acknowledge. For instance, in Figure 2a the user is required to click on a "Next" button that will take the user to the next stage of the phishing UX flow (the data stealing phase). This is referred to as *click-through* cloaking in [45], because this type of UX design not only tends to lend a sense of legitimacy to the website, but also hinders the ability of existing web security crawlers to explore internal pages. Similarly, some phishing victims may be presented with different types of CAPTCHAs that must be solved or a one-time-password (OTP) that must be entered before proceeding, as shown in Figure 5. Other sites that elicit login credentials pretend that the first login attempt failed and ask for the credentials a second time, likely as a way to verify whether a user is actually typing the credentials and responding correctly to the error message by entering them again.

Challenges: The challenge in this case is to proceed beyond the first page, to be able to explore the remaining stages of the phishing UX. Admittedly, this is difficult, especially in the presence of CAPTCHAs and OTPs. Although there exist previous works on automatic CAPTCHA breaking [21, 42], it is very hard to break the large variety of CAPTCHAs, including custom ones, used on phishing websites. Rather than attempting to break all different CAPTCHAs, we focus on automatically recognizing and measuring what type of *click-through* or CAPTCHA is presented to the user and if it relies on a well-known user verification library or custom code.

3 METHODOLOGY OVERVIEW

We now provide an overview of our methodology for automatically interacting with phishing websites and collecting information that we will use to analyze the UX and UI design patterns they employ. An overview of our measurement framework is shown in Figure 6

Overall, our approach to measuring phishing UX and UI patterns can be divided into two main parts: (i) an intelligent web crawler



(a) (b)

Figure 5: Example of (a) CAPTCHA and (b) OTP on a modern phishing website.

(Section 4) that is designed to visit a phishing website, infer what type of inputs the site elicits from users, simulate user interactions (including entering forged but syntactically valid data), and navigate through the website's pages as a victim would do; and (ii) a data analysis component (Section 5) that uses the data collected by the crawler to measure UX and UI characteristics of modern phishing websites.

Intelligent Phishing Crawler: We develop a custom crawler that can automatically interact with phishing websites. To this end, we developed a set of custom Puppeteer [18] scripts to extract all input elements, perform JS event logging, automate page analysis to infer what the web page is requesting, and decide how to interact with the page next. To extract input elements, the crawler first identifies them by parsing the web page's DOM tree, computes the visual location (i.e., the rendering bounding box) of these elements, and then uses a set of heuristics that combine DOM analysis and OCR to find the text associated with each input box. Next, once the crawler derives all input fields and their respective textual descriptions, it uses a statistical classifier that is trained to map each input textual description to an *input type*. Finally, the crawler leverages Faker [1] to forge syntactically valid data for each input type, enters this data into the form and submits it to the phishing site.

If no input box is found on a page, as in the case of a *click-through* page (see example in Figure 2a), the crawler attempts to interact with other page elements, such as hyperlinks and buttons. To identify the presence of buttons on a page, the crawler uses a combination of DOM analysis and a deep learning-based object detector that we trained specifically to identify buttons on web pages. The crawler stops when no more progress can be made (e.g., no new data can be submitted and the crawler is not able to advance to a next page). More technical details can be found in Section 4.

UX and UI Data Analysis: As our crawler navigates through a phishing website, it collects detailed information about the pages it encounters, which we use to analyze the UX and UI design patterns used in modern phishing websites. We broadly categorize the design patterns we study as User Interface (UI) Patterns, Multi-Stage

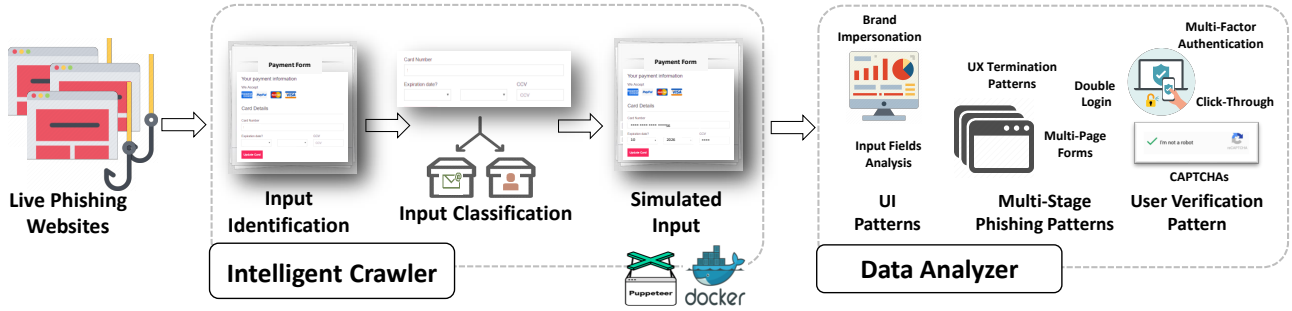


Figure 6: Overview of our intelligent crawler and data analysis process.

Phishing Patterns and User Verification Patterns. Under UI Patterns, we include measurements related to the visual design of the pages and to characteristics of the data stealing forms. For instance, we measure the occurrence frequency with which different data types are requested from the victim. Under the Multi-Stage Phishing category, we measure how frequently phishing websites use multiple steps to collect user data, and report the distribution of the data types that are requested at each step. In addition, we also measure other UX patterns that are associated with multi-step phishing pages, such as patterns related to how the victim's experience on the phishing website ends. These UX termination patterns include redirecting the victim to an error page, to the login page of the impersonated brand's legitimate website, to a "congratulations" or "success" message, or to a message that reassures the user that her data is safe.

We also separately measure how many phishing sites employ user verification techniques such as click-through, CAPTCHAs and multi-factor authentication. During our study, we noticed that phishing sites use a variety of methods to accomplish this goal. Especially, different phishing sites that make use of CAPTCHAs implement different CAPTCHA types. This makes it difficult to automatically recognize whether a CAPTCHA is present on a web page or not. To solve this problem, we built a custom object detection system that is specifically trained to recognize many different types of CAPTCHAs embedded in web pages. We describe our technical approach in more details in Section 5.

4 INTELLIGENT PHISHING CRAWLER

As mentioned in Section 3, our crawler is designed to visit a phishing website, interpret what the website is asking for by using a combination of page DOM analysis and machine learning-based computer vision methods, forge and submit syntactically correct data into the phishing site's forms, and overall navigate the phishing experience end-to-end, as a real victim would do. Along the way, the crawler collects a variety of information about the pages it encounters to enable the UX and UI analysis described in Section 5.

We built our intelligent crawler on top of Puppeteer [18] and the latest stable Chrome browser. To enable at-scale measurements, we built a Docker-ized version of our crawler, and used a "clean" container for each phishing website we visit, to take advantage of a fresh browser profile for each browsing session. We then created a *crawler farm* that allowed us to visit and collect data from more than 1,000 phishing websites per day, on average.

Given a URL selected from our phishing URLs feed, the crawler automatically loads the corresponding web page and waits until the browser has rendered it and the network is idle (e.g., using Puppeteer's `networkidle2` option). It then captures a screenshot of the page and the current state of the DOM tree for analysis. In the following, we describe how the crawler interprets the content of a page based on the state of the DOM and visual page rendering, and how it decides what actions to take next.

4.1 Input Field Identification

First, the crawler attempts to identify whether the page contains any form elements, such as input boxes and drop-down menus. To this end it explores the DOM tree looking for `input` and `select` nodes (for brevity, in the following when referring to `input` we mean either `input` or `select`). For each of these DOM elements, the crawler does the following:

(1) Collect related DOM elements, such as the `form` element that an `input` belongs to and their node properties such as `id`, `name`, `type`, `placeholder text` and `innerText` (if present).
(2) Collect information about neighboring DOM elements (i.e., parent and sibling nodes), including nodes such as `div`, `span`, text contained under those nodes, etc., which may provide information about what type of data an `input` field may be asking for.
(3) Because phishing pages are sometimes structured to make DOM analysis difficult (see example in Figure 3), we also perform a visual analysis of the elements around each `input` box. Specifically, we aim to identify neighboring text labels that may explain what the input field is requesting. To this end, we first compute the rendering bounding box around the `input` field (e.g., see Listing 1 in Appendix), and then by using the Tesseract OCR engine [8] on visual regions of the page to the left and on top of the input box (up to a threshold distance, measured in pixels).

Once the input fields on a page are identified and information about their neighboring page elements and text labels has been collected, we send this information to an *input field classifier*, as described below.

4.2 Input Field Classifier

The goal of the input field classifier is to take the information collected by the input field identification module described earlier and to infer the type of data that each `input` box is expecting. To this

end, we build a multi-class classifier that, for each `input` box, takes the related information as input, translates it into a feature vector, and outputs a *data type* label selected among a predetermined set of common types, including *name*, *email*, *password*, *phone number*, *credit card number*, etc. (a complete list of data type labels is shown in Table 6 in appendix). To achieve this goal we take the following steps:

(1) *Feature vector computation*: Given DOM nodes and node properties related to an `input` box collected by the input field identification module (Section 4.1), we first extract all strings included in the node's content and node property values, as well as the labels extracted from the visual rendering of the page via OCR. We then filter out noisy strings by removing stop-words and possible special characters (e.g., non-ASCII characters), and by only keeping valid dictionary words (including common acronyms). We then use a bag-of-words approach to compute feature vectors.
(2) *Data type labeling*: Given the features related to an `input` box, we pass them to a multi-class classifier based on the *SGDClassifier* algorithm [6], which performed better on this task than other common statistical classifiers (e.g., *Multinomial Naive Bayes*). The classifier then outputs the data type label with the highest confidence score. To filter out possible classification errors, we set a conservative threshold of 0.8 on the data type confidence score. Data type labels that have a lower than 0.8 confidence score are rejected and the input box is labeled as *unknown*.
(3) *Training the classifier*: Training is performed iteratively, in an *active learning* setting. The classifier is initially trained on a relatively small dataset of manually labeled input fields. Then, the classifier is used to attribute a label to new input fields found on new phishing websites. Input fields that are labeled as *unknown* are sent to a human expert to be labeled manually, and then fed back to the classifier for re-training. To reduce manual effort and help experts label unknown input fields, we developed a custom web application that presents all the information gathered about each input field to the user and highlights the input box on the related web page screenshot in which it was found. This enables the expert to visually infer the data type related to the input box and thus to record the correct ground truth label.

Model Training and Evaluation. To train and test this model, we used our crawler to collect text extracted (including via OCR) from input fields found in real-world phishing sites and formed a custom dataset made of these texts. We then manually assigned input field categories by referring to the related webpage's screenshots. Overall, we labelled 1,310 input field samples. Of these, we used 1,000 samples for training and tested the model on the remaining 310 samples. To evaluate the performance of this multi-class input field classifier, we calculated the F1-score metric across each of the labels and averaged all F1-score values, resulting in an average score of 90%. Table 6 (in Appendix) presents a detailed breakdown of the F1-score, Precision and Recall per input field category.

4.3 Simulating and Submitting Input Data

Once the `input` boxes have been labeled, we leverage Faker [1] to forge syntactically correct data for each input field. The data type labels output by the input field classifier map directly into data types available in the Faker library, making it straightforward to forge the corresponding data. For input fields labeled as *unknown* the crawler enters a predetermined default string. Finally, we need to submit the form. However, this step is less straightforward than it may seem at first. This is because the crawler needs to automatically identify how to trigger the form submission, which can be complicated by the adversarial web design patterns used by phishing websites.

We attempt to submit the forged data by trying the following approaches:

- *Enter*: the crawler simulates the `Enter` key press while the focus is on an input box.
- *Submit button*: we perform DOM analysis to check whether the DOM contains a button. This includes checking for DOM elements such as `button` and `input` with a type attribute being `image` or `submit`. If such an element is found, the crawler clicks on it. In addition, if these standard elements are not found, we also look for hyperlinks that may be styled as a button. To identify these, we apply a set of heuristics for DOM analysis.
- *Form action*: if we cannot find a button via DOM analysis, for instance, because the page visually renders the submit button using HTML "tricks", such as using `canvas`, SVG `path`, etc., we look for a `form` element in the DOM, get a JavaScript object reference, `e`, that points to it (e.g., via its `id` or `name`, or by traversing the DOM), and invoke `e.submit()`.
- *Visual detection*: we also attempt to identify a button by building a deep-learning object detection module that is trained specifically to detect buttons in web pages. To this end, we start with a pre-trained Faster-RCNN [34], and fine-tune it using 10,000 screenshots of automatically generated webpages with a variety of randomly selected logos, input boxes, CAPTCHAs, and labeled buttons. Because the placement of the buttons in these pages is known, it can be used for fine-tuning the Faster-RCNN model.

After each attempt to submit the data, we check whether the browser transitioned to a new page. Notice that the phishing site's main interest is to steal users' data, and it is not easy for the attackers to verify if the data is real or forged in real time, as long as it is syntactically correct. Therefore, if the user is presented with a different page after submission, this typically indicates that the data has been accepted and sent to the attacker's server. On the other hand, if the crawler detects that no page transition occurred, it will attempt to generate a new set of forged data and try to submit again. This is needed because some of Faker's data types result in generating syntactically valid but randomly chosen data, and we noticed that in some cases the data that is generated is not accepted as valid by some phishing forms. To address this issue, we attempt to submit forged data on a given page for up to three times, before aborting the browsing session. Notice also that detecting whether the page indeed changed is not entirely straightforward, as it is not sufficient to check whether the URL changed. To this

end, we employ the page transition detection method described in the following section.

4.4 Page Transitions and Termination

If no input box is found, the crawler still attempts to find a button-like element to interact with, using the same approach described earlier to detect a form submission button. This is motivated by the fact that some phishing websites may present the user with a first "click-through" button, like the one in the example in Figure 2a. If no element is found with which to interact, the crawler stops making progress (i.e., no new page is reached) and the browsing session will therefore terminate. On the other hand, the crawler will continue to interact with the website if a new page is detected, until no more progress is made or a timeout (20 minutes, in our experiments) is reached.

To determine if the crawler is making progress after interacting with a page, we devised a set of heuristics to determine whether the page changed. For instance, after submitting data on a page such as the one in Figure 2d, the user may be presented with a different form, like in Figure 2e or some kind of termination page, like in Figure 2f. In either of the these two scenario, the page changes, and thus we determine that the crawler has made progress and repeat the same process described earlier on the newly reached page.

Determining if the page changes is easy when the browser navigates to a new URL. However, this is not always the case. In practice, we found that it is not uncommon for the page to visually change while the URL remain the same. This can is achieved via JavaScript code that dynamically changes the content of the page. In this latter case, we still want to automatically detect a page change and determine that the crawler has been making progress (i.e., we do not want to prematurely end our crawling session). To solve this issue, every time we load a page we compute a *lightweight DOM hash*. Specifically, we traverse the DOM tree (depth-first) and keep only `input`, `div`, `span`, `button`, and `label` elements, which are often sufficient to "shape" the structure of a phishing page. We then concatenate the HTML tag of these nodes (in depth-first order) and compute a hash of the resulting string. After the crawler interacts with the page, it recomputes the DOM hash of the page and checks for changes. If the hash differs, the crawler infers that the page has visually changed (even if they URL remained the same). Other more sophisticated approaches for page change detection are possible, but we found the use of this approach to be efficient and effective in practice.

4.5 Metadata Collection

Throughout the entire crawling session, the crawler collects a large variety of information about the pages it encounters. First, it will keep a record of all the DOM and visual analysis it performed, including information about all input elements, buttons, etc. that it encountered. In addition, the crawler also captures information about all network requests made by the page, including any redirections and the content of the network responses. Concurrently, the crawler also instruments the JavaScript code running on each page it visits to record all calls made by the page's code to the `addEventListener` API and all JavaScript events that are triggered while a page is rendered. We use these network and JavaScript activity logs in our post-processing analysis, for instance to measure how many phishing pages may use a keylogger to steal users' data even if they happen to realize (a bit late) that they may be interacting with a malicious page and decide not to press the submit button (see Section 5.1).

4.6 Crawler Farm Setup

To automatically crawl phishing sites at scale, we leverage Docker containers [17] to launch several parallel instances of our intelligent crawler. To run this crawler farm, we used an Ubuntu 16.04 Linux machine with 128 GB of memory and a total of 30 Docker sessions in parallel at a time. We set a timeout of 20 minutes for each of the phishing site crawling sessions. Overall, we visited *56,027* phishing websites in a period of 43 days between March 20th, 2022 and May 1st, 2022.

Table 1: Summary of crawling results (count of unique URLs and domains)

# OpenPhish Seed URLs	# Filtered Phishing URLs	# Crawled Phishing URLs	# Crawled Phishing SLDs
56,027	51,859	66,072	25,693

Live Phishing Feed: We obtain our seed phishing URLs from OpenPhish [7], one of the largest commercial-grade repositories of phishing websites (we subscribed to their premium data feed). The feed is updated with new live phishing websites every 5 minutes. To visit the newly reported URLs, we immediately spawn new crawler instances and point them to the new phishing URLs.

To filter out possible noise from the URL feed, we also check the URLs against a phishing detection product from a leading security vendor. As shown in Table 1, we started with a total of *56,027* phishing URLs, and after filtering we were left with *51,859* confirmed phishing URLs. Using perceptual hashing, in a way similar to previous work [37], we clustered the first page of each of these phishing websites to identify phishing campaigns; namely, we group together phishing websites that use the same brand and UI but are hosted under different domain names. In this way, we discovered *8,472* distinct phishing campaigns targeting over *381* brands. Of these, *3,214* campaigns consisted of less than 50 phishing sites each, and only 11 campaigns included more than 500 phishing sites, thus providing a widely diverse set of phishing websites and targeted brands for analysis. The top 10 targeted brands in our dataset are shown in Table 7 (in Appendix). Table 2 shows the top 10 business categories that were targeted by phishing sites as obtained from the "Industry sector" information provided by the OpenPhish premium feed.

5 PHISHING UX AND UI ANALYSIS

In this section, we dig deeper into the approaches we used to measure each of the User Interface design and User Experience design observed in modern phishing websites and present the results of our analysis on UI and UX based characteristics of modern phishing sites.

Table 2: Top business categories targeted by phishing sites in our dataset

Business Category	Count of Phishing Sites
Online/Cloud Service	10,057
Financial	10,053
Social Networking	5,268
Logistics & Couriers	3,985
Email Provider	2,177
Cryptocurrency	2150
Telecommunications	1,408
e-Commerce	1,271
Payment Service	1,154
Gaming	657

5.1 UI Patterns

We start by analyzing characteristics of the visual layout and design of the pages, including characteristics of the data stealing input boxes.

5.1.1 Brand Impersonation vs. Cloning. In order to evade visual-based phishing detectors, phishing sites that impersonate legitimate websites do not necessarily need to clone their design, as discussed in Section 2. To measure the number of phishing websites that make this design choice, we leverage VisualPhishNet [11], which provides a deep learning model trained to detect phishing sites that closely resemble a set of legitimate sites' visual design. We also use the brand information provided with the phishing URL in the OpenPhish premium feed as ground truth, and input the screenshot of those phishing pages into VisualPhishNet's model.

If the model is unable to correctly identify the legitimate website targeted by the phishing page, we infer that the design of the phishing site do not closely mimic the design of the impersonated legitimate site. For instance, the phishing website shown in the example Figure 1a (Section 2) shows a phishing page that impersonates the DHL brand. However, the page is misclassified by VisualPhishNet as "Alibaba" rather than DHL (perhaps due to the use of similar colors for the two brands). This is due to the fact that the phishing page does not closely mimic the actual DHL website (Figure 1b).

Table 3: Results showing the percentage of phishing sites that do not clone visual design of Top 5 brands

Brand	Microsoft OneDrive	Facebook, Inc	DHL Airways, Inc	Chase Personal Banking	Netflix
% of Sites	58%	84%	30%	12%	26%

Results. Given that the brand labels used to train VisualPhishNet and those provided by OpenPhish are not consistent with each other, reconciling the labels required manual effort. Therefore, we performed an analysis on a subset of 250 phishing websites impersonating 5 different popular brands that we found in our dataset. For each brand, we randomly selected 50 screenshots related to different phishing campaigns that targeted that specific brand (with a roughly equal number of screenshots per campaign). Table 3 shows the percentage of web page screenshots that did not closely mimic their legitimate brand counterparts. On average, 42% of the cases were related to campaigns that do not clone the targeted brand's visual appearance. This further indicates the need to improve vision-based phishing detection with additional context-based information.

5.1.2 Input Fields Distribution. Besides visual appearance, we also measure two additional UI characteristics: (i) the distribution of data types requested by phishing pages, and (ii) in how many cases phishing sites use an obfuscated page design that requires the use of OCR to be able to infer the data type associated to the input boxes (see Section 4.1). To measure these traits, we leverage the logs collected by our crawler regarding the type of input field it identifies and if OCR was applied to identify these types.

Results. Figure 7 shows the distribution of input fields that were found in real-world phishing pages. Besides categorizing the input fields into commonly requested field types requested by phishing pages, we further grouped the filed types into higher-level categories, or context groups, including *Login*, *Personal*, *Social*, and *Financial* information. As it can be seen, *Email* and *Password* are the most requested information, in *28,736* and *35,762* pages respectively. At the same time, our analysis found evidence of several other types of user information being commonly requested by phishing websites as well.

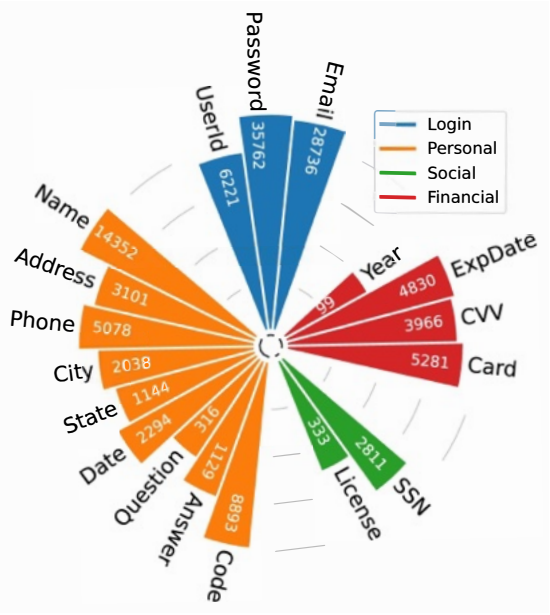


Figure 7: Plot displaying distribution of input field categories and their context groups across all phishing pages

In addition, we found that our crawler had to revert to using OCR in 27% of the cases, in which it could not identify any useful input field information via DOM element analysis. This indicates that some type of UI obfuscation was likely used to hinder web security crawlers from easily identifying the inputs requested by the page. Also, in 12% of the cases, no "standard" submit button was found in the page, and our crawler had to submit a form or transition to the next page by interacting with button coordinates identified via visual object detection (see Section 5.3.2).

5.1.3 Keylogging. Typically, a legitimate web page will send sensitive user data to the server only after the user explicitly submits it (e.g., by pressing `Enter`, clicking on the submit button, etc.). However, phishing sites may obviously benefit from sending sensitive information to the attacker's server as soon as the user enters the data into a phishing page. This enables the website to steal (partial) information even if the user realizes midway that the site may be unsafe to use. We measure in how many cases this design pattern is implemented by checking how many phishing pages implement *keylogger* behavior by using JS instrumentation to actively monitor triggered *keydown* events.

Results. Our analysis revealed that 18,745 phishing sites were monitoring *keydown* events using a listener and stored the data as they were typed. Further, 642 of these sites made a network request immediately after data was entered into the input element, and we confirmed that 75 among these sites sent a request that included the data that was entered into the input field before any sort of explicit submit action was performed. This demonstrates how some attackers aim to steal the data as soon as it is entered, rather than risk losing the data by waiting until the user submits the form (e.g., in case the user realizes the risk and decides not to click on the submit button).

5.2 Multi-Stage Phishing Patterns

As shown in the examples in Section 2, modern phishing websites may include a multi-stage phishing attack that includes (i) a user verification stage, (ii) multiple data stealing pages, and (iii) a termination page that congratulates or reassures the user that their data has been correctly submitted and is safe. In this section we discuss how we measure aspects related to the latter two characteristics and their measurements result (user verification patterns are discussed in Section 5.3).

5.2.1 Multi-Page Web Forms. As discussed earlier, many phishing websites do not only focus on harvesting users' login credentials. Phishing sites that request additional or different information (e.g., financial data) tend to do so by mimicking users' experience on legitimate websites using multiple web forms/pages to request different types of data. To measure how many phishing sites use this approach, we analyze the data collected by our intelligent crawler (Section 4) and determine how many phishing websites required the crawler to input different types of forged data on different web pages.

Results. Of the *51,859* sites that were crawled, we found that *23,446* (45%) of the phishing sites required the crawler to input different types of data at different stages of the attack. More notably, we observed that some phishing sites employed up to 5 steps (i.e., 5 different pages) to steal user information. We measured the distribution of phishing sites with a total of n pages, for n ranging from 2 to 5 (see Figure 8) and found, for instance, that over *12,000* of these sites included 3 stages of phishing. Next, we also measured the distribution of input field types across these multiple steps. Figure 9 displays the counts associated with input field types that were requested at different stages of the multi-phishing sites. It can be observed that login information was vastly requested in the first two stages as compared to personal information, which

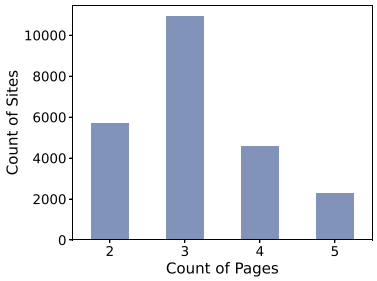


Figure 8: Plots displaying a histogram of total page count for multi-step phishing sites

was instead requested more often in the later stages of phishing, compared to the initial page. Similar trends can be observed for social and financial data, which are frequently requested in the middle stages of multi-stage phishing attacks. Note that the reason for a relatively high occurrence of social and financial information also in the first stage of the attacks may be due in part to our live phishing feed pointing our crawler to start from an "internal" URL of a multi-step phishing site, rather than its initial true page.

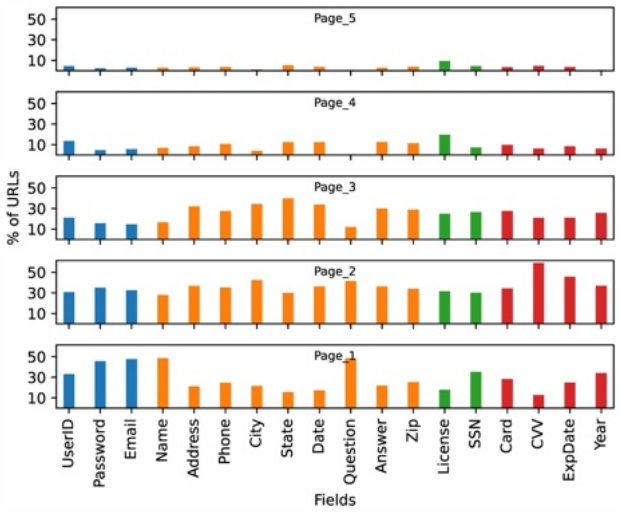


Figure 9: Plots displaying the Count of Field Categories found at different pages of the Phishing sites with multi-step phishing. The percentage of URLs is calculated per field type to indicate the distribution of that field across multiple steps and not compared to other field types.

5.2.2 Double Login. Some phishing pages use a design pattern referred to as "Double Login," wherein once a user enters the login credentials for the first time, the site requests the user to enter the credentials again by pretending that the entered information was incorrect. This happens regardless of the validity of the information the user enters in the first page and appears to be used by phishing sites to verify the information given by the users [5] and also to

remove any suspicion from the user in case she intentionally entered wrong information the first time (and it may also thwart security crawlers). This "feature" is also advertised as a major selling point in a number of phishing kits (e.g., see example in Figure 12, in Appendix). In our measurements, we identify such cases of "Double Login" by starting from the phishing sites that were found to use a multi-page data stealing pattern. Then, we check if any of these sites presented the crawler with two consecutive pages that both asked for the same login credentials (e.g., data types such as username, email, password, phone number, etc.).

Results. By applying a set of heuristics, such as finding two consecutive pages that asked for the same set of login-related information, we discovered a total of 400 phishing sites that used a *Double Login* pattern.

5.2.3 UX Termination Patterns. Another aspect of phishing that needs more attention is how the experience typically ends for the victims. Interestingly, many modern phishing websites are designed to present the user with a final reassuring message. Presumably, this is done to prevent the user from realizing her information was stolen and take immediate steps to change her credentials, block credit cards, freeze her account, etc. Also, this may be a way to prevent the phishing URL from being reported to security vendors.

To measure what type of phishing terminal page users may encounter, we first manually analyze a set randomly sampled terminal pages collected by our crawler. Specifically, we consider only phishing websites that exhibit a multi-page phishing pattern (as defined earlier) and whose last page visited by our intelligent crawler did not contain any input box (i.e., the terminal page did not request and additional data from the user). Finally, we consider only the last page visited by the crawler under the same domain name as the domain of the initial phishing URL the crawler started from. Among the remaining pages, we manually label 300 randomly chosen pages under four different categories: *Success Message*, *Custom Error Message*, *HTTP Error*, and *Phishing Awareness*. Given this labeled dataset, we then train a bag-of-words text classifier that can automatically parse a web page, extract its text, and categorize it among one of the four categories listed above. Our analysis led to the following observations.

Results. While some phishing sites opt to terminate the UX by showing again the same (e.g., last visited) page again to the user, we observed two unique termination patterns. In the first of these patterns, the site would redirect the user to a legitimate website, which often coincides with the legitimate brand website targeted by the phishing attack or with another popular websites such as google.com. Table 4 displays the top legitimate effective second-level domains that users would be redirected to at the end of the attack. Overall, 7,258 distinct phishing sites were found to navigate to a total of 680 distinct legitimate domains.

In the second pattern, a number of sites were found to display a custom message to the user at the end of the attack. To measure at scale the type of terminal messages displayed by phishing sites, we first manually labeled 200 randomly selected samples and divided them into 4 categories: *success message*, *custom error message*, *HTTP error*, and *phishing awareness*. We then trained a machine learning classifier on these 200 labeled samples and tested it on a separate set of 100 manually labeled samples. The test results showed that our multi-class classifier achieved 97% accuracy (we also implemented a reject option, whereby test samples that fell below a detection threshold on the maximum class confidence score of less than 0.65 were discarded). Overall, *5,403* of the multi-stage phishing sites were found to display a final page with no input fields present. Among those, 966 were related to success messages, 125 to error messages, 1,599 resulted in HTTP errors and 176 to fake phishing awareness/training messages, as shown in the example in Figure 4. Considering the sites that displayed phishing awareness/training messages, we found that they could be grouped into 41 unique campaigns displaying different messages related to phishing attack simulations. These messages appear to be designed to reassure the users that they were *almost* phished, their data is safe and they need not worry. Unfortunately, based on our crawler logs we could observe that the data entered by the crawler in the previous steps of the attack was indeed sent to the attacker's server.

Table 4: Top Benign Effective SLDs that were found in Terminal Navigation Pattern

Second-level Domain	Count	Second-level Domain	Count
microsoftonline.com	459	google.com	133
dhl.com	297	godaddy.com	118
glacierbank.com	249	citi.com	109
office.com	219	bt.com	96
microsoft.com	218	americafirst.com	92
example.org	197	youtube.com	85
example.net	189	chase.com	76
mtb.com	188	yahoo.com	70
example.com	184	alaskausa.org	61
live.com	180	netflix.com	47

5.3 User Verification Patterns

In this section, we focus on measuring user verification patterns, namely cases in which a phishing website presents the user with some kind of simple challenge, before giving them access to the pages containing the data stealing forms. As discussed earlier, this design pattern can provide two benefits to the attackers: i) lend a sense of legitimacy to the phishing website by mimicking user verification patterns also used on legitimate sites, and ii) thwarting web security crawlers.

Obviously, our intelligent crawler is also hindered by some of these user verification patterns, such as CAPTCHAs. For instance, given the large variety of possible different CAPTCHA types that may be used by phishing sites, it would be very difficult to build a generic system that automatically identifies their presence, type, and also then automatically solve the identified CAPTCHA type. Although there exist previous works that focus on solving specific types of CAPTCHAs, to our knowledge no universal solver that can break arbitrary CAPTCHA types has been proposed. Rather than attempting to break CAPTCHAs or other user verification techniques, we instead focus on measuring how many phishing sites implement user verification and what type of verification patterns they deploy, as described below.

5.3.1 Click-Through. To measure the occurrence of click-through pages, we start by considering only those phishing websites that

Figure 10: Examples of CAPTCHAs found on recent phishing sites.

were found to have a multi-stage pattern. Then, we select those websites where a page visited by the crawler does not include any input boxes, but the subsequently visited page does. This indicates cases in which the crawler was able to identify an element to interact with in the initial page (e.g., a button) that allowed it to successfully navigate to the initial data-stealing page.

Results. Phishing websites that use a click-through pattern typically implement a multi-stage attack. Therefore, we measure the click-through patterns only across such sites. As a result, we found that 2,933 out of the 23,446 multi-stage phishing sites included a click-through pattern. This accounts for 5.6% of the total number of phishing sites that we analyzed. Furthermore, 2,713 of these were found on the first page encountered by the crawler for a given site, while 220 of them were found in the internal pages (past the first one) of multi-stage phishing sites.

5.3.2 CAPTCHAs. To measure whether a phishing website makes use of CAPTCHAs and of what type, we follow these steps. To this end, we first attempt to determine if the website uses a popular JavaScript library for known CAPTCHAs, such as Google's ReCaptcha [10], Hcaptcha [9], etc. We refer to this broad category as "known CAPTCHAs." Then, we also attempt to determine if a phishing website includes a "custom CAPTCHA," namely a real CAPTCHA that does not rely on a specific popular CAPTCHA library. This latter task is clearly not straightforward because it requires us to visually detect the presence of a CAPTCHA challenge, which in turn can be one of many different types.

To visually detect the presence of a CAPTCHA, we leverage a deep learning-based approach. Specifically, we use a pre-trained Faster R-CNN [34] object detection model from Facebook's *Detectron2* library [4], and fine-tune it with a large training dataset of web pages that we generated containing CAPTCHAs. To this end, we first collected an extensive and diverse collection of CAPTCHA challenge images from a publicly available CAPTCHA dataset [40] (see examples in Figure 10). Then, we use a large collection of brand logos provided by Phishpedia [24] to automatically generate a set of web pages that contain a logo, a CAPTCHA challenge image, an input box and a submit button (in case of simulated text-based CAPTCHA). Some examples of CAPTCHA pages we generated are shown in Figure 13, in Appendix. Given that for each generated web page we know the exact location of elements including CAPTCHAs, we can create a dataset with the label of the element and its bounding box. We then use this labeled dataset to fine-tune the Faster R-CNN object detection model. For the hyper-parameters, we set the base learning rate, "BASE_LR", to 0.001, the maximum iterations parameter, "MAX_ITER" to 3000 and "BATCH_SIZE_PER_IMAGE" parameter as 64.

Table 5: Captcha Detection Model results reporting Average Precision(AP) per class for testing set

	Text-based CAPTCHAS					
	Type 1	Type 2	Type 3	Type 4	Type 5	Type 6
Count	315	315	315	315	315	315
Average Precision	91.0	99.4	98.9	95.8	97.5	98.5
	Visual CAPTCHAs			**Button**	**Logo**	
	Type 1		Type 2			
Count	420		315	1324	1370	
Average Precision	80.7		92.1	89.2	77.1	

Model Evaluation: To evaluate the model that detects custom captchas, we first train it on 10,000 artificially generated images that contain a mix of different types of CAPTCHAs, logos and buttons as explained above. Then, for validation and testing, we used a different set of 1,000 and 2,000 artificially generated web pages, respectively. Our experiments resulted in an average precision of 91.9% and 92.0% for the validation set and test set, respectively. Table 5 shows the per-class precision obtained for the test set.

Given that the model did well on the test set, we then proceed to test the model on a set of real-world phishing images obtained from crawling during our experimentation phase. On providing images to the model, it outputs the bounding box coordinates of any CAPTCHAs, buttons or logos that it detects. Unlike the test images that we generated, real-world images are not annotated. Therefore, we manually inspected the results to verify the model's prediction. Considering that websites can be styled in multiple ways, to avoid confusion due to noise generated by additional page elements, we additionally apply the following heuristics to filter misclassified CAPTCHA elements.

(1) When the model detects a text-based CAPTCHA, we leverage the bounding boxes outputted by the model and check if there is an input field located next to the detected CAPTCHA. If no such input field was found in its vicinity, then we consider those as misclassified. In the case that there was an input field, we ensure that that crawler hasn't already mapped it to any other valid category such as name, email, etc.
(2) In case of visual captchas, we slice the part of the image identified as visual captcha using the given bounding box and the input image. Next, we determine if this portion of the image is similar to at least 3 of visual captchas that exist in our training dataset. To calculate similarity, we compare the distance between pHash of the prediction portion of image and pHashes of corresponding class's image in the dataset. If there were at least 3 instances where the distance calculated is below a threshold (set to 20 in our case), only then we consider the prediction valid.

We had collected 7,814 unique images from our crawling. After labelling these images manually, we found 123 of these images to have some form of CAPTCHAs (visual or text-based). On feeding these images to the model, we initially obtained a precision of 89.2% and recall of 87.8%. After applying the heuristics-based filtering, we were able to discard all false positives, since they did not satisfy the rules described above. As a result, our model's precision increased to 100% while the recall remained unchanged at 87.8%, thus denoting

that our model missed few cases of captchas that are significantly dissimilar to ones that were seen during training. However, to improve this, the model could be retrained to detect new forms of CAPTCHA.

Results. Based on the information collected by our crawler, we can measure the prevalence of both *known* and *custom* CAPTCHAs on phishing websites. We distinguish between three common types of CAPTCHAs: text-based, visual-based or invisible (i.e., behavior-based). Considering all 51,859 phishing websites we crawled, we found evidence of CAPTCHA deployment in 2,608 of them. Among these, we found that 2,496 phishing sites contained JavaScript code related to known CAPTCHAs, such as Google Recaptcha (1,856 sites) and hCaptcha (640 sites). Interestingly, phishing sites that include third-party CAPTCHA libraries may expose themselves to being indexed and crawled by the library's provider (e.g., Google), and thus risk early detection by web security companies.

We also noticed that in most cases in which known CAPTCHA libraries were used, they provided visual-based or behavior-based CAPTCHAs. At the same time, text-based CAPTCHAs appeared to be mostly custom CAPTCHA schemes. Therefore, to more accurately identify custom CAPTCHAs, we used our object detection model. Overall, our object detector identified 34 text-based custom CAPTCHAs and 78 visual-based custom CAPTCHAs.

5.3.3 Multi-Factor Authentication. Recently, some phishing websites have started to request users to enter a two-factor authentication (2FA) code sent via email or text message. This strategy is used for instance by MITM phishing toolkits [23]. In this study, we do not attempt to study how MTIM phishing works, as done in previous work [23]. Rather, we are concerned only with measuring how many phishing websites request a form of 2FA. To find such cases, we focus our attention on phishing pages that, according to our intelligent crawler, ask for a *code* data type. Then, perform a post-processing analysis on the logs collected by the crawler from those pages and search for pages whose input box labels include a keyword related to 2FA (we compile a set of common keywords based on manual analysis of sample 2FA pages).

Figure 2 shows an example of modern phishing website that request a 2FA code. After our intelligent crawler navigated correctly through the first few pages (including two click-through pages and different data-stealing pages) it is presented with a request for a "one-time code." It turns out that this was a *fake* 2FA. In fact, our crawler identified in real time that the input box was requesting a code, and simply generated and submitted a random sequence of numbers (using the Faker library), and it was redirected to the terminal "success" page.

Results. To measure if this trait is used in any of the analyzed sites, we first filter those pages containing "Code" input fields. Next, we analyze those fields and select those that contains keywords related to two-factor authentication, such as 'OTP', 'SMS', '2FA', etc. As it can be seen from Figure 7, there were 8,893 sites that contained one or more input fields identified as "Code." Among those, we found 1,032 of them that requested a one-time authentication code (via SMS). A few examples are shown in Figure 14 (in appendix).

6 DISCUSSION OF LIMITATIONS

As noted in [26], simply using features such as visual similarity to a legitimate web page or brand logo detection may not be sufficient for accurate phishing website detection. Instead, the authors propose to combine brand similarity with automatic identification of credential-taking (or stealing) intentions. However, login credentials are not the only input types that phishing sites request from users, as shown in our measurement results. In general, the ability to inspect a web page and automatically understand what it is asking for, as done by our intelligent crawler, may represent an additional powerful feature to further boost the accuracy of phishing detection systems such as [26].

Another possibility is to embed a system similar to our crawler in the browser, to "test" suspected phishing pages in real time. For instance, assume a user visits a web page that is classified (e.g., using a system such as VisualPhishNet [11] or Phishpedia [24]) as suspicious; namely, a possible phishing attack. Rather than immediately raising a potentially false alert, the browser could allow the user to interact with the page. However, if the page includes input fields and the user starts entering data into the form, the browser could temporarily buffer this data without immediately passing it to the page. At the same time, in the background the browser could use a system similar to our intelligent crawler to interact with the page, investigate the UI/UX, and determine whether this is actually a phishing website. If yes, the user will be alerted, and since the data was buffered by the browser and not passed to the page, the user's information will be safe. If not (i.e., the page is determined to be benign), the browser could reload the original page the user started to interact with and transparently enter the previously buffered user data. Obviously, such a system would need to minimize latency in obtaining the final classification result, to avoid impacting usability. However, users are relatively slow to enter data, and the detection system could operate in the background while the user is allowed to interact with a buffered page. While potentially promising, these ideas would require significant browser engineering efforts and therefore, we leave them to future research.

Perfectly mimicking the behavior of a real user that interacts with a phishing website is obviously a very challenging task. In particular, it is very difficult for an automated system to interpret what a web page may be asking the user to do, especially when the system can encounter an unbounded variety of websites and be faced with adversarial web design patterns. Nonetheless, our intelligent crawler and data analysis approximate user interpretation of, and interaction with, web pages by using a combination of page DOM analysis, visual analysis (e.g., OCR), deep learning-based computer vision and heuristics informed by in-depth domain knowledge of how phishing pages are constructed. Obviously, our classifiers and heuristics are not perfect, and thus they can make mistake that impact the exact measurement results. For instance, we noticed that our CAPTCHA object detection model failed to identify a number of visual-based CAPTCHA instances generated by the hCaptcha library. After analyzing the misclassified cases, we found that the missed CAPTCHAs looked all alike and contained a dark variant of hCaptcha that was not encountered during training data collection and labeling. This points to the fact that additional effort is needed to collect and label training data for the crawler's machine learning

components. However, it is important to notice that, as shown in our evaluation, the classifiers we built are sufficiently accurate to be used in practice. And while our measurements cannot be 100% accurate, we are confident that they accurately capture the "big picture" trends and characteristics of modern phishing websites. Ultimately, we believe that the findings from our study can be used to significantly strengthen future defenses, including state-of-the-art phishing detectors such as [26].

Another limitation of our measurement framework is due to the fact that we currently focus only on phishing websites that use the English language. However, with more engineering effort our tools can be extended to support different languages. For instance, this would include training the input field classifier with input text labels from languages other than English. We plan to explore this extension of our current framework in future work. Also, our crawler may further benefit from directly learning how real users interact with web pages. For instance, it may be possible to perform a user study in which users are asked to browse both legitimate and phishing-like websites, to record the set of behaviors (mouse movements, web page components they interact with, etc.) they exhibit. This dataset of user behaviors could then be used to train more advanced models that can imitate human actions in a more comprehensive way.

One important obstacle for our crawler is represented by phishing websites that use man-in-the-middle (MITM) phishing kits [23]. In this case, the user (and thus the crawler) would be presented with content fetched from the legitimate website targeted by the phishing attack. Therefore, the crawler would need to use valid login credentials to move forward. In addition, the crawler may also need to bypass a multi-factor authentication code, such as a one-time password (OTP) sent via SMS. These *transparent phishing* attacks and attacks that require OTPs are outside the scope of our crawler, as more specialized measurement and defense systems would be needed in such cases [23].

7 ETHICAL CONSIDERATIONS

To perform our study, we automatically interacted with tens of thousands of malicious websites, including submitting forged data. These websites are all reported as *phishing* by OpenPhish (openphish.com), a commercial-grade phishing URL feed. In addition, to remove potential noise from the URL feed we leveraged a commercial-grade phishing detection system provided by a leading cybersecurity company. In addition, although a small amount of noisy (legitimate) URLs in our feed may be unavoidable, our crawler does not cause any harm to the websites it interacts with, besides submitting syntactically correct but invalid user information.

8 RELATED WORK

The problem of phishing has been researched for more than two decades. Over the years, several of the works have been focused on detecting phishing attacks with different techniques such as visual similarity [11, 13, 20, 46] and machine learning [25, 27, 44]. More recently, multiple works [14, 15, 26, 38, 39, 41, 43] have focused on using deep learning to classify phishing pages with higher accuracy. In terms of practical client-based defenses against phishing, blocklisting [2, 35] remains the de-facto front line defense. Blocklists, as well as the aforementioned detectors, typically rely upon on web security crawlers to gather data, thus prompting security researchers to evaluate them for attacks [12, 28, 30, 31]. The intelligent crawler system we developed in this paper can help improve the coverage of these security crawlers by enabling them to interact with the phishing sites like a potential victim would.

Some recent works have also begun to look at measuring and understanding in-the-wild phishing attacks from different perspectives such as phishing kits [16, 19, 29], man-in-the-middle phishing techniques [23], client-side cloaking techniques [45], credential stealing mechanisms [33]. In this paper, we analyzed in-the-wild phishing pages from a different vantage point by capturing the end-to-end experience of victims on a large number of real world phishing sites. It is to be noted that [33] also attempted to automatically interact with phishing sites with the help of some heurisitcs and OCR techniques to automatically supply credentials. However, their work was limited to analyzing the data transfer of only the login credentials which is only a small part of the experience that a phishing victim can be potentially subject to. On the other hand, our work is focused on automatically recreating the complete virtual experience of a victim for each site, thereby gaining a deep understanding of all information they attempt to steal from victims as well as the UI experiences that the victims are subject to. Besides serving as a measurement tool, this system can also be useful in enabling new kinds of mitigation measures by enhancing prior works such as [32]. For example, the loading of a third-party's HTTP resources (such as a bank) on the UX termination page of a phishing site can be used as an alarm signal for our proposed crawler to visit it. If our crawler observes that the site steals social security numbers, the third-party organization can immediately forward the victim's IP address to government agencies for pursuing mitigation measures.

9 CONCLUSION

In this paper, we proposed a novel methodology to study phishing at scale by combining browser automation with machine learning to simulate user interactions with phishing pages and explore their UX and UI characteristics. Using our intelligent crawler, we were able to explore over 50,000 phishing websites and automatically identify a number of visual and UX trends used by modern phishing sites to lend their site an air of legitimacy. Accordingly, we found that many phishing sites take users through multiple phishing stages to steal more than just their login credentials, and leverage a number of mechanisms to evade security crawlers while making the site believable to the end users. The measurement results we presented can further aid in the development of more accurate and robust phishing detectors.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers and our shepherd, for their constructive comments and suggestions on how to improve this paper. This material is based in part upon work supported by the National Science Foundation (NSF) under grants No. CNS-2126641 and CNS-2126655. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

REFERENCES

[1] 2014. Faker: Library docuemntation. https://faker.readthedocs.io/en/master/.

[2] 2020. Google Safe Browsing : Blocklisting Platform. https://safebrowsing.google.com/.

[3] 2021. CISCO :2021 Cyber Security Trends. https://learn-umbrella.cisco.com/ebook-library/2021-cyber-security-threat-trends-phishing-crypto-top-the-list. (Last accessed Sep 19, 2022).

[4] 2021. Detectron2. https://github.com/facebookresearch/detectron2.

[5] 2021. Don't Get CAPTCHA'd By This New Phishing Technique! https://firstcallhelp.tamu.edu/2021/09/dont-get-captchad-by-this-new-phishing-technique/. (Last accessed Sep 19, 2022).

[6] 2021. SKlearn: SGDClassifier. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html.

[7] 2022. OpenPhish: Phishing Intelligence. https://openphish.com/.

[8] 2022. Pytesseract Package. https://pypi.org/project/pytesseract/.

[9] 2022. What is hCaptcha? https://www.hcaptcha.com/what-is-hcaptcha-about.

[10] 2022. What is ReCaptcha? https://developers.google.com/recaptcha.

[11] Sahar Abdelnabi, Katharina Krombholz, and Mario Fritz. 2020. VisualPhishNet: Zero-Day Phishing Website Detection by Visual Similarity. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 1681–1698. https://doi.org/10.1145/3372297.3417233

[12] Bhupendra Acharya and Phani Vadrevu. 2021. PhishPrint: Evading Phishing Detection Crawlers by Prior Profiling. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 3775–3792. https://www.usenix.org/conference/usenixsecurity21/presentation/acharya

[13] Sadia Afroz and Rachel Greenstadt. 2011. PhishZoo: Detecting Phishing Websites by Looking at Them. In *Proceedings of the 2011 IEEE Fifth International Conference on Semantic Computing (ICSC '11)*. IEEE Computer Society, USA, 368–375. https://doi.org/10.1109/ICSC.2011.52

[14] S. Bagui, D. Nandi, S. Bagui, and R. J. White. 2019. Classifying Phishing Email Using Machine Learning and Deep Learning. In *2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*. 1–2.

[15] Eduardo Benavides, Walter Fuertes, Sandra Sanchez, and Manuel Sanchez. 2020. Classification of Phishing Attack Solutions by Employing Deep Learning Techniques: A Systematic Literature Review. In *Developments and Advances in Defense and Security*. Springer, 51–64.

[16] Hugo Bijmans, Tim Booij, Anneke Schwedersky, Aria Nedgabat, and Rolf van Wegberg. 2021. Catching Phishers By Their Bait: Investigating the Dutch Phishing Landscape through Phishing Kit Detection. In *Proceedings of the 30th USENIX Security Symposium*. USENIX Association, 3757–3774.

[17] Docker. 2019. Docker: Enterprise Container Platform. https://www.docker.com/. (Last accessed Nov.1, 2019).

[18] Google. 2019. Puppeteer: Chormium Browser Automation Tool. http://liwc.wpengine.com/compare-dictionaries/. (Last accessed Nov.11, 2019).

[19] Xiao Han, Nizar Kheir, and Davide Balzarotti. 2016. PhishEye: Live Monitoring of Sandboxed Phishing Kits. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 1402–1413. https://doi.org/10.1145/2976749.2978330

[20] M. Hara, A. Yamada, and Y. Miyake. 2009. Visual similarity-based phishing detection without victim site information. In *2009 IEEE Symposium on Computational Intelligence in Cyber Security*. 30–36.

[21] Imran Hossen, Yazhou Tu, Md Fazle Rabby, Nazmul Islam, Hui Cao, and Xiali Hei. 2020. An Object Detection based Solver for Google's Image reCAPTCHA v2. In *RAID*.

[22] M. Khonji, Y. Iraqi, and A. Jones. 2013. Phishing Detection: A Literature Survey. *IEEE Communications Surveys Tutorials* 15, 4 (2013), 2091–2121.

[23] Brian Kondracki, Babak Amin Azad, Oleksii Starov, and Nick Nikiforakis. 2021. Catching Transparent Phish: Analyzing and Detecting MITM Phishing Toolkits *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 36–50. https://doi.org/10.1145/3460120.3484765

[24] Yun Lin, Ruofan Liu, Dinil Mon Divakaran, Jun Yang Ng, Qing Zhou Chan, Yiwen Lu, Yuxuan Si, Fan Zhang, and Jin Song Dong. 2021. Phishpedia: A Hybrid Deep Learning Based Approach to Visually Identify Phishing Webpages. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 3793–3810. https://www.usenix.org/conference/usenixsecurity21/presentation/lin

[25] Gang Liu, Bite Qiu, and Liu Wenyin. 2010. Automatic Detection of Phishing Target from Phishing Webpage. In *Proceedings of the 2010 20th International Conference on Pattern Recognition (ICPR '10)*. IEEE Computer Society, USA, 4153–4156. https://doi.org/10.1109/ICPR.2010.1010

[26] Ruofan Liu, Yun Lin, Xianglin Yang, Siang Hwee Ng, Dinil Divakaran, and Jin Song Dong. 2022. Inferring Phishing Intention via Webpage Appearance and Dynamics: A Deep Vision Based Approach. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*.

[27] S. Marchal, K. Saari, N. Singh, and N. Asokan. 2016. Know Your Phish: Novel Techniques for Detecting Phishing Sites and Their Targets. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. 323–333.

[28] Sourena Maroofi, Maciej Korczyński, and Andrzej Duda. 2020. Are You Human? Resilience of Phishing Detection to Evasion Techniques Based on Human Verification. In *Proceedings of the ACM Internet Measurement Conference* (Virtual Event, USA) *(IMC '20)*. Association for Computing Machinery, New York, NY, USA, 78–86. https://doi.org/10.1145/3419394.3423632

[29] Adam Oest, Yeganeh Safaei, Adam Doupé, Gail-Joon Ahn, Brad Wardman, and Gary Warner. 2018. Inside a phisher's mind: Understanding the anti-phishing ecosystem through phishing kit analysis. *2018 APWG Symposium on Electronic Crime Research (eCrime)* (2018), 1–12.

[30] Adam Oest, Yeganeh Safaei, Adam Doupé, Gail-Joon Ahn, Brad Wardman, and Kevin Tyers. 2019. PhishFarm: A Scalable Framework for Measuring the Effectiveness of Evasion Techniques against Browser Phishing Blacklists. In *2019 IEEE Symposium on Security and Privacy (SP)*. 1344–1361. https://doi.org/10.1109/SP.2019.00049

[31] Adam Oest, Yeganeh Safaei, Penghui Zhang, Brad Wardman, Kevin Tyers, Yan Shoshitaishvili, and Adam Doupé. 2020. PhishTime: Continuous Longitudinal Measurement of the Effectiveness of Anti-phishing Blacklists. In *USENIX Security Symposium*.

[32] Adam Oest, Penghui Zhang, Brad Wardman, Eric Nunes, Jakub Burgis, Ali Zand, Kurt Thomas, Adam Doupé, and Gail-Joon Ahn. 2020. Sunrise to Sunset: Analyzing the End-to-end Life Cycle and Effectiveness of Phishing Attacks at Scale. In *USENIX Security Symposium*.

[33] Peng Peng, Chao Xu, Luke Quinn, Hang Hu, Bimal Viswanath, and Gang Wang. 2019. What Happens After You Leak Your Password: Understanding Credential Sharing on Phishing Sites. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security* (Auckland, New Zealand) *(Asia CCS '19)*. Association for Computing Machinery, New York, NY, USA, 181–192. https://doi.org/10.1145/3321705.3329818

[34] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems* 28 (2015).

[35] Steve Sheng, Brad Wardman, Gary Warner, Lorrie Cranor, Jason Hong, and Chengshan Zhang. 2009. An empirical analysis of phishing blacklists. (2009).

[36] Kurt Thomas, Frank Li, Ali Zand, Jacob Barrett, Juri Ranieri, Luca Invernizzi, Yarik Markov, Oxana Comanescu, Vijay Eranti, Angelika Moscicki, Daniel Margolis, Vern Paxson, and Elie Bursztein. 2017. Data Breaches, Phishing, or Malware? Understanding the Risks of Stolen Credentials. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 1421–1434. https://doi.org/10.1145/3133956.3134067

[37] Phani Vadrevu and Roberto Perdisci. 2019. What You See is NOT What You Get: Discovering and Tracking Social Engineering Attack Campaigns. In *Proceedings of the Internet Measurement Conference* (Amsterdam, Netherlands) *(IMC '19)*. Association for Computing Machinery, New York, NY, USA, 308–321. https://doi.org/10.1145/3355369.3355600

[38] Grega Vrbančič, Iztok Fister, and Vili Podgorelec. 2018. Swarm Intelligence Approaches for Parameter Setting of Deep Learning Neural Network: Case Study on Phishing Websites Classification. In *Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics* (Novi Sad, Serbia) *(WIMS '18)*. Association for Computing Machinery, New York, NY, USA, Article 9, 8 pages. https://doi.org/10.1145/3227609.3227655

[39] Bo Wei, Rebeen Ali Hamad, Longzhi Yang, Xuan He, Hao Wang, Bin Gao, and Wai Lok Woo. 2019. A Deep-Learning-Driven Light-Weight Phishing Detection Sensor. *Sensors* 19, 19 (2019), 4258.

[40] Rodrigo Wilhelmy and Horacio Rosas. 2013. *captcha dataset*.

[41] P. Yang, G. Zhao, and P. Zeng. 2019. Phishing Website Detection Based on Multidimensional Features Driven by Deep Learning. *IEEE Access* 7 (2019), 15196–15209.

[42] Guixin Ye, Zhanyong Tang, Dingyi Fang, Zhanxing Zhu, Yansong Feng, Pengfei Xu, Xiaojiang Chen, and Zheng Wang. 2018. Yet Another Text Captcha Solver: A Generative Adversarial Network Based Approach. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 332–348. https://doi.org/10.1145/3243734.3243754

[43] Ping Yi, Yuxiang Guan, Futai Zou, Yao Yao, Wei Wang, and Ting Zhu. 2018. Web phishing detection using a deep learning framework. *Wireless Communications and Mobile Computing* 2018 (2018).

[44] Haijun Zhang, Gang Liu, Tommy W. S. Chow, and Wenyin Liu. 2011. Textual and Visual Content-Based Anti-Phishing: A Bayesian Approach. *Trans. Neur. Netw.* 22, 10 (Oct. 2011), 1532–1546. https://doi.org/10.1109/TNN.2011.2161999

[45] Penghui Zhang, Adam Oest, Haehyun Cho, Zhibo Sun, RC Johnson, Brad Wardman, Shaown Sarker, Alexandros Kapravelos, Tiffany Bao, Ruoyu Wang, Yan Shoshitaishvili, Adam Doupé, and Gail-Joon Ahn. 2021. CrawlPhish: Large-scale Analysis of Client-side Cloaking Techniques in Phishing. *2021 IEEE Symposium on Security and Privacy (SP)* (2021), 1109–1124.

[46] Yu Zhou, Yongzheng Zhang, Jun Xiao, Yipeng Wang, and Weiyao Lin. 2014. Visual Similarity Based Anti-Phishing with the Combination of Local and Global

Table 6: Field Classifier results reporting Precision, Recall and F1-score. The last row reports the average F1-score across all categories.

Field Category	Precision	Recall	F1-Score	Count
Login Data				
Email	0.92	1	0.95	23
UserId	0.71	0.83	0.76	6
Password	0.97	0.94	0.95	36
Personal Data				
Name	0.92	0.90	0.91	52
Address	1	0.88	0.94	18
Phone	0.95	1	0.97	23
City	0.91	0.91	0.91	12
State	0.8	1	0.88	5
Question	1	1	1	10
Answer	1	1	1	14
Date	0.77	0.7	0.73	10
Code	1	0.95	0.97	21
Social Data				
License	1	0.6	0.8	5
SSN	0.81	0.81	0.81	11
Financial Data				
Card	0.85	0.92	0.88	25
ExpDate	0.9	1	0.94	18
CVV	0.9	0.69	0.78	13
Other Data				
Search	1	0.87	0.93	8
Overall results			0.90	310

Figure 12: Screenshot of a Description of a Real Phishing Kit with Double Login Feature

Table 7: Top Brands that were targeted by Phishing Sites in our dataset

Targeted Brand	Count of Phishing Sites
Office365	5,351
DHL Airways, Inc.	3,069
Facebook, Inc.	2,335
WhatsApp	2,257
Tencent	1,701
Crypto/Wallet	1,687
Outlook	1,437
La Banque Postale	1,131
Chase Personal Banking	1,071
M & T Bank Corporation	1,015

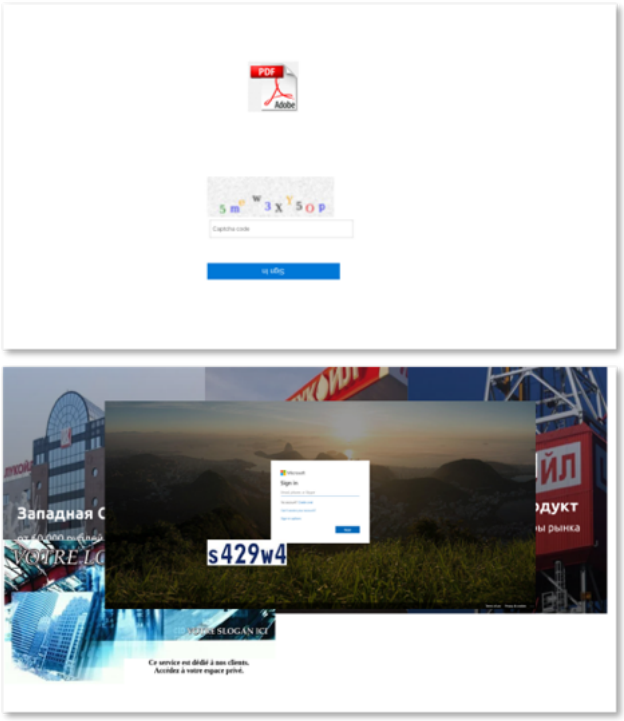


Figure 13: Examples of Images that were generated to train Object DEtection Model

Features. In *Proceedings of the 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications (TRUSTCOM '14)*. IEEE Computer Society, USA, 189–196. https://doi.org/10.1109/TrustCom.2014.28

A ADDITIONAL EXAMPLES

```
    function get_elements(){
        var recs = [];
        var e = document.getElementsByTagName('*');
        for (var i=0; i<e.length; i++) {
            var rect = e[i].getBoundingClientRect();
            recs[i]={}
            if (rect != undefined){
                recs[i].right = rect.right;
                recs[i].top = rect.top;
                recs[i].bottom = rect.bottom;
                recs[i].left = rect.left
                recs[i].height = rect.height
            }
            recs[i].innerText  = e[i].textContent ;
            recs[i].tag        = e[i].tagName!= undefined? e[i
].tagName : '';
            recs[i].id         = e[i].id!=null?e[i].id:'None';
            recs[i].name       = e[i].name!=null?e[i].name:'
None';
            recs[i].type       = e[i].type ;
            recs[i].text       = e[i].Text ;
            recs[i].value      = e[i].value ;
            recs[i].visibility = e[i].style!= undefined ? e[i
].style.visibility : 'null' ;
            recs[i].placeholder= e[i].placeholder!=null?e[i].
placeholder!=''?e[i].placeholder:'null':'null';
        }
        return recs;
    }
```

Listing 1: Template for count-based policies

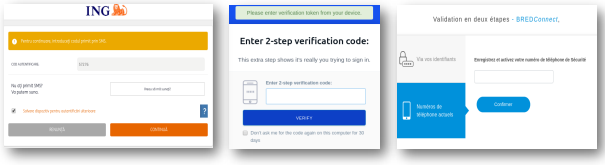


Figure 14: Few Examples of Phishing pages that employed two-factor authentication and requested for SMS code.

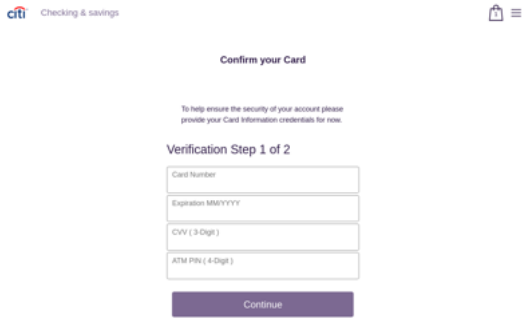


Figure 11: Example of phishing website that does not require login credentials.