CrawlPhish: Large-scale Analysis of Client-side Cloaking Techniques in Phishing

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*§
*Arizona State University, †PayPal, Inc., ‡North Carolina State University, §Samsung Research
*{penghui.zhang, aoest, haehyun, zhibo.sun, tbao, fishw, yans, doupe, gahn}@asu.edu
†{raouljohnson, bwardman}@paypal.com, ‡{ssarker, akaprav}@ncsu.edu

***Abstract*—Phishing is a critical threat to Internet users. Although an extensive ecosystem serves to protect users, phishing websites are growing in sophistication, and they can slip past the ecosystem's detection systems—and subsequently cause real-world damage—with the help of evasion techniques. Sophisticated *client-side* evasion techniques, known as *cloaking*, leverage JavaScript to enable complex interactions between potential victims and the phishing website, and can thus be particularly effective in slowing or entirely preventing automated mitigations. Yet, neither the prevalence nor the impact of client-side cloaking has been studied.**

In this paper, we present CrawlPhish, a framework for automatically detecting and categorizing client-side cloaking used by known phishing websites. We deploy CrawlPhish over 14 months between 2018 and 2019 to collect and thoroughly analyze a dataset of 112,005 phishing websites in the wild. By adapting state-of-the-art static and dynamic code analysis, we find that 35,067 of these websites have 1,128 distinct implementations of client-side cloaking techniques. Moreover, we find that attackers' use of cloaking grew from 23.32% initially to 33.70% by the end of our data collection period. Detection of cloaking by our framework exhibited low false-positive and false-negative rates of 1.45% and 1.75%, respectively. We analyze the semantics of the techniques we detected and propose a taxonomy of eight types of evasion across three high-level categories: *User Interaction*, *Fingerprinting*, and *Bot Behavior*.

Using 150 artificial phishing websites, we empirically show that each category of evasion technique is effective in avoiding browser-based phishing detection (a key ecosystem defense). Additionally, through a user study, we verify that the techniques generally do not discourage victim visits. Therefore, we propose ways in which our methodology can be used to not only improve the ecosystem's ability to mitigate phishing websites with client-side cloaking, but also continuously identify emerging cloaking techniques as they are launched by attackers.

I. Introduction

Despite extensive research by the security community, phishing attacks remain profitable to attackers and continue to cause substantial damage not only to the victim users that they target, but also the organizations they impersonate [27, 55]. In recent years, phishing websites have taken the place of malware websites as the most prevalent web-based threat [22, 52]. Even though technical countermeasures effectively mitigate web-based malware, phishing websites continue to grow in sophistication and successfully slip past modern defenses [46].

In a cat-and-mouse game with the anti-phishing ecosystem, sophisticated phishing websites implement evasion techniques to delay or avoid detection by automated anti-phishing systems, which, in turn, maximizes the attackers' return-on-investment [43]. Such evasion—known as *cloaking*—typically seeks to determine if a visitor to the website is a bot, and shows benign content if so. The danger posed by successful evasion is exacerbated by these websites' efforts to steal more than just usernames and passwords: today's phishing attacks seek to harvest victims' full identities, which can cause wider damage throughout the ecosystem and is more challenging to effectively mitigate [54].

Thwarting phishers' evasion efforts is, thus, an important problem within the anti-phishing community, as timely detection is the key to successful mitigation. Prior research has characterized server-side cloaking techniques used by phishing websites [30, 37, 44] and showed that they can defeat key ecosystem defenses such as browser-based detection [43]. However, the nature and prevalence of advanced cloaking techniques, such as those implemented on the *client-side* using JavaScript, is poorly understood. Client-side cloaking can be particularly dangerous because it enables the implementation of complex interactions with potential victims.

By analyzing—at scale—the client-side source code of known phishing websites in the wild, we can not only gain an understanding of the evasion techniques used by phishers, but also leverage this understanding to improve existing phishing detection systems and guide the mitigations used by the ecosystem. Unlike server-side code used by phishing websites, client-side code can trivially be obtained through web crawling. However, a key challenge in gaining further insights from this data is the dynamic nature of JavaScript code, which hampers automated analysis [32]. In this paper, we overcome this challenge and evaluate client-side evasion by developing *CrawlPhish*.

CrawlPhish is a robust framework that harvests the source code of live, previously reported phishing websites in the wild and automatically detects and categorizes the client-side cloaking techniques used by these websites. By efficiently adapting advanced program analysis techniques inspired by prior research of JavaScript malware [18, 32, 34, 36], our framework can not only identify the semantics of these cloaking techniques, but also track the evolution of code written by specific phishing kit authors [16].

We use the CrawlPhish framework to perform a large-scale

evaluation of the landscape of client-side cloaking used by phishing websites. In total, over a period of 14 months from mid-2018 to mid-2019, we collected and thoroughly analyzed 112,005 phishing websites. We measured the prevalence of client-side cloaking techniques within these websites and discovered that 35,067 (31.3%) use such cloaking. Thereof, we identified 1,128 groups of related implementations which we believe stem from distinct threat actors. Moreover, we observed that the percentage of phishing websites with client-side cloaking grew from 23.32% in 2018 to 33.70% in 2019.

To understand why client-side cloaking is used so frequently, we characterize the ways in which it functions, and we define eight different types of evasion techniques in three high-level categories: *User Interaction*, *Fingerprinting*, and *Bot Behavior*. Respectively, the techniques within these categories require human visitors to perform a task, profile the visitor based on various attributes, or exploit technical differences between browsers used by crawlers and real browsers.

We evaluated CrawlPhish and found that it could detect the presence of cloaking with low false-positive (1.45%) and false-negative (1.75%) rates, while requiring an average of 29.96 seconds to analyze each phishing website. Once CrawlPhish has detected cloaking, it can then reliably categorize the *semantics* of the cloaking technique by using both static and dynamic code features.

Finally, to show that client-side cloaking poses a real-world threat, we deploy 150 carefully-controlled artificial phishing websites to empirically demonstrate that all three categories of evasion can successfully bypass browser-based detection by Google Chrome, Microsoft Edge, and other major web browsers. We also demonstrate that these websites remain accessible to potential human victims. As a result, we disclosed our findings to the aforementioned browser developers, who are working to improve the timeliness of the detection of the corresponding phishing websites.

Our analysis furthers the understanding of the nature of sophisticated phishing websites. In addition, the CrawlPhish framework can be deployed to continuously monitor trends within complex evasion techniques while identifying new types of techniques as they are introduced by attackers. Our methodology can not only directly help address gaps in the ecosystem's detection of sophisticated phishing websites, but can also aid in the development of attributes to improve existing anti-phishing mitigations such as browser-based detection. Our contributions are thus as follows:

- A scalable, automated framework for evaluating client-side evasion techniques used by phishing websites in the wild, supported by a novel adaptation of multiple JavaScript code analysis approaches.
- The first in-depth study of the nature and prevalence of client-side evasion techniques used by sophisticated phishing websites, and a taxonomy of these techniques based on semantic categorization.
- Measurements indicating the increasing use of client-side evasion techniques by phishers, and an empirical evaluation showing that these techniques represent a threat to the current ecosystem.
- Methodology for improving the ability of ecosystem anti-phishing defenses to detect highly evasive phishing websites.

Cloaking Type	Attributes	Examples
Server-side	HTTP Request	Repeat Cloaking IP Cloaking User-agent Cloaking Referrer Cloaking
Client-side	Client-side Characteristics Execution of JavaScript	Redirection Cloaking

TABLE I: Summary of cloaking types from previous studies.

II. Background

Over the past years, a myriad of techniques have been implemented by the anti-phishing ecosystem to detect and mitigate phishing attacks [44]. Analysis of phishing URLs [11, 12, 29, 33] and website content [10, 13, 62, 64, 67] has given rise to ecosystem-level defenses such as e-mail spam filters, malicious infrastructure detection, and URL blacklists.

Specifically, systems such as Google Safe Browsing [61] and Microsoft SmartScreen [40] power the anti-phishing backends that display prominent warnings across major web browsers when phishing is detected. These warnings are primarily blacklist-based: they rely on content-based detection. Evasion techniques commonly used by phishing websites are capable of bypassing or delaying such blacklisting [38, 43, 45].

A. Cloaking Techniques in Phishing

Attackers leverage *cloaking techniques* to evade detection by anti-phishing systems: phishing websites with cloaking display benign-looking content instead of the phishing page whenever they suspect that a visit originates from security infrastructure [44]. Cloaking techniques can be categorized into two groups: server-side and client-side (Table I shows examples of each type). Server-side cloaking techniques identify users via information in HTTP requests [59]. Client-side cloaking is implemented through code that runs in the visitor's browser (JavaScript) to apply filters using attributes such as cookies or mouse movement.

Existing anti-cloaking methodologies focus on bypassing server-side cloaking by comparing the visual and textual features of different versions of a crawled website retrieved by sending multiple web requests with different configurations (e.g., user agents or IP addresses) [25, 30, 59]. Client-side cloaking techniques, however, are still poorly understood due to challenges in automatically analyzing JavaScript code and understanding its semantics. Moreover, neither the prevalence nor impact of client-side cloaking has been investigated in the context of phishing.

Figure 1 shows how client-side cloaking techniques are used in phishing websites. Cloaking code embedded in the HTTP response payload shows different web page content based on the identification of visitors (as either potential victims or

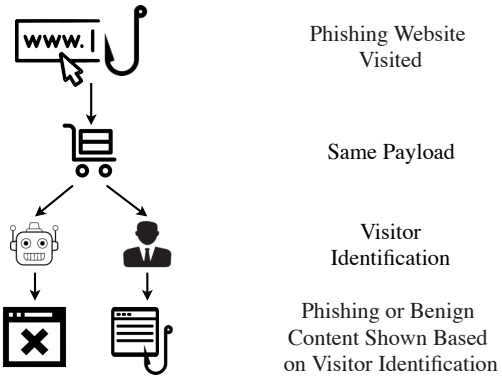


Fig. 1: Typical operation of client-side cloaking in phishing websites.

bots). Consequently, cloaked phishing websites may have a longer life span than ones without: by delaying or avoiding detection, the attackers who launch these websites maximize their return-on-investment [43]. Because client-side evasion techniques enable complex interactions between potential victims and phishing websites, they may be more effective in hampering automated detection than traditional server-side cloaking, and, thus, pose a threat to potential victim users.

B. Challenges in Analyzing Client-side Cloaking

Unlike server-side code, the client-side code (JavaScript) of websites can trivially be obtained through crawling. Therefore, malicious websites typically leverage code *obfuscation* methods such as string array encoding, object key transformation, dead code injection, and even full encryption [17, 31]. Attackers also can dynamically generate and execute code (e.g., using `eval`) to hide malicious behaviors. Such obfuscation methods pose a challenge for static code analysis approaches, which are otherwise favored for their efficiency.

Other types of obfuscation also seek to prevent dynamic analysis approaches from detecting malicious behaviors. Malicious JavaScript code often targets specific versions of web browsers and operating systems by *fingerprinting* them [18]. Such attacks are difficult to discover because detection systems require extensive resources to reveal the conditions that trigger attacks [17]. Besides, external and inter-block *dependencies*, which require recording states in different execution paths, can be obstacles that thwart the analysis of JavaScript code [34]. Furthermore, scripts may execute in an *event-driven* manner to necessitate external triggers to initiate malicious behavior while otherwise appearing benign [34].

All of the aforementioned anti-analysis methods can potentially be leveraged by phishing websites' implementations of client-side cloaking techniques. Given the difficulty of analyzing such cloaking, the security community struggles to thoroughly understand the impact and prevalence of phishers' tactics, and, thus, may fail to appropriately mitigate them. When we consider the scale on which phishing attacks occur [9], the consequences of the corresponding gaps in detection and mitigation can be significant.

Cloaking Category	Cloaking type	Requirement
User Interaction	Pop-up	Click on alert/notification window
	Mouse Detection	Move mouse over browser
	Click Through	Pass Click Through on browser
Fingerprinting	Cookie	Check document.cookie
	Referrer	Check document.referrer
	User-Agent	Check navigator.userAgent
Bot Behavior	Timing	Render webpage after certain time using sleep()/Date.getTime()
	Randomization	Show content randomly using Math.random()

TABLE II: Summary of the client-side cloaking technique types identified in this work.

III. Overview

Client-side cloaking techniques can help phishing websites evade detection by anti-phishing entities [43], yet prior studies have not investigated them in detail, despite evidence that sophisticated phishing websites—such as those with client-side cloaking—are responsible for a majority of real-world damage due to phishing [46].

We discover eight different types of JavaScript cloaking techniques across three high-level categories: User Interaction, Fingerprinting, and Bot Behavior (summarized in Table II). Cloaking techniques in the *User Interaction* category show phishing content only if visitors interact with a phishing website (e.g., by moving the mouse or clicking a specific button). Phishing websites with *Fingerprinting* identify visitors by inspecting the configuration of browsers or web requests. Finally, phishing websites with *Bot Detection* identify anti-phishing crawlers based on factors such as how long the web page stays open and whether the web request is repeated after failing initially. We elaborate on each cloaking type in Section VI-A.

We aim to comprehensively understand and characterize the landscape of client-side cloaking techniques used by phishing websites in the wild through an automated methodology for analyzing them. To this end, we design, implement, and evaluate *CrawlPhish*: a framework that automatically detects and analyzes client-side cloaking within phishing websites. Figure 2 provides an overview of the CrawlPhish architecture. CrawlPhish is composed of the following components:

① **Crawling and pre-processing (§IV-A):** CrawlPhish first collects web page source code (along with any external file inclusions) by visiting live phishing website URLs recently reported to anti-phishing feeds. We then filter URLs that cannot be retrieved as well as URLs without any JavaScript code.

② **Feature extraction (§IV-B):** CrawlPhish adapts a state-of-the-art code analysis method, *forced execution* [34], to execute JavaScript regardless of branch conditions, and extracts all possible execution paths in which evasion techniques could be implemented. We then derive (1) visual features of the rendered web pages, by means of screenshots, and (2) code structure features such as web

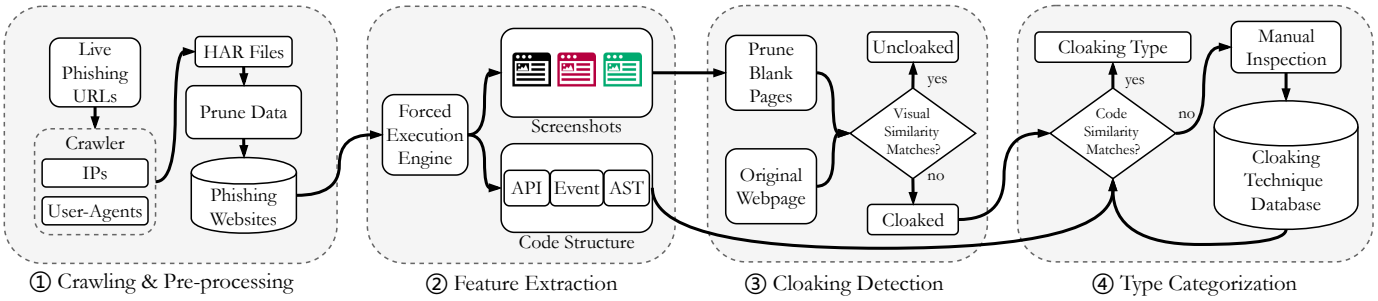


Fig. 2: CrawlPhish architecture.

API calls, event listeners, and the Abstract Syntax Tree (AST) for each path.

③ **Cloaking detection (§IV-C):** CrawlPhish analyzes the visual features corresponding to each execution path to detect if cloaking exists, and it stores the corresponding code structure features of every such path.

④ **Cloaking categorization (§IV-D):** Using the code structure features, CrawlPhish categorizes the cloaking techniques used by phishing websites based on their semantics.

After presenting CrawlPhish and the resulting analysis of cloaking techniques, we evaluate our approach, as described below, to ensure that our methodology can help improve user security by enhancing the ability of anti-phishing systems to detect and bypass attackers' evasion techniques.

§V. Detection of cloaked phishing websites: We first evaluate the effectiveness of CrawlPhish on the dataset of 112,005 phishing websites that we crawled. We show that CrawlPhish can detect the presence of client-side cloaking with very low false-negative and false-positive rates (1.75% and 1.45%, respectively).

§VI. Cloaking categorization: We measure the prevalence of client-side cloaking techniques *in the wild* and characterize eight different types in three high-level categories. Also, we evaluate CrawlPhish to show that it can reliably categorize the *semantics* of each cloaking technique. We compare the findings from our crawled dataset with an additional dataset of 100,000 phishing websites. Moreover, we analyze the source code that CrawlPhish collected to identify and group related cloaking implementations. Tracking the deployment and evolution of such code can be indicative of sophisticated phishing kits, which can help security researchers pinpoint the threat actor and track the associated attack volume.

§VII. Impact of cloaking techniques: We deploy 150 artificial phishing websites to empirically demonstrate that all three categories of evasion can successfully bypass detection by the anti-phishing backends used in major web browsers. Separately, we conduct a user study to show that human users remain likely to interact with cloaked phishing pages. Through these experiments, we show that client-side cloaking poses a real-world threat.

Dataset. In our evaluation, we use two different datasets. (1) *APWG Dataset*: CrawlPhish collected the source code of 28,973 phishing websites from June to December 2018 and 100,000 websites from May to November 2019 using the Anti-Phishing Working Group (APWG) URL feed [51].
(2) *Public Dataset*: Phishing website source code from September to December 2019 from various well-known sources, shared publicly by a security researcher [24].

Ethics. We ensured that our experiments did not cause any disruption to legitimate Internet infrastructure or negatively impact any human users. Our crawling (Section IV-A) did not negatively affect any legitimate websites because CrawlPhish pruned those websites before initiating analysis. The user study in Section VII-B underwent the IRB review and received approval. During this study, we did not ask for or acquire any Personally Identifiable Information (PII) from participants. In addition, no human users ever saw any of the artificial phishing websites discussed in Section VII-A, nor were these websites configured to collect any data that may have been submitted.

IV. CrawlPhish Design

The design goal of CrawlPhish is to detect and categorize client-side cloaking techniques in an automated manner while overcoming the JavaScript code analysis challenges discussed in Section II-B.

A. Crawling & Pre-processing

To collect the source code of *live* phishing websites to detect and classify client-side evasion methods that are currently employed in the wild, CrawlPhish first obtains URLs of known phishing websites in real-time.

In our deployment, CrawlPhish continuously ingested URLs from the APWG eCrime Exchange database—a curated clearinghouse of phishing URLs maintained by various organizations engaged in anti-phishing. Because this database receives frequent updates and tracks phishing URLs that target a diverse range of brands, it is well-suited for phishing website analysis.[1] Note, however, that the inclusion of a URL in the database does not mean that it was adequately mitigated (e.g.,

[1] Although the goal of cloaking is to evade detection by automated anti-phishing systems, such evasion will often delay detection rather than outright prevent it. Phishing websites may also be detected by other means (e.g., manual review) [46]. Thus, we expected the AWPG database to contain a representative sampling of any client-side cloaking that might be used in the wild.

through timely blacklisting) [45]. Hence, websites found to use sophisticated client-side cloaking still warrant scrutiny.

Next, CrawlPhish downloads source code by visiting each phishing website URL (shortly after being ingested) using a programmatically controlled web browser. Specifically, CrawlPhish stores source code using HAR files [57], which capture all HTTP requests/responses between our client and the server, and ensure that all dependencies (such as linked scripts) are preserved for each website. In case of a failed request, CrawlPhish switches between different configurations of IP addresses and user-agents in an effort to circumvent potential server-side cloaking techniques used by phishing websites [44]. 4,823 of the 128,973 websites we crawled (3.74%) showed different response status codes after we switched request configurations.

Finally, CrawlPhish filters out URLs that contain blank pages or non-phishing websites. Such websites were either already taken down [7] or were false-positive detections by the time of crawling. We found 0.53% of URLs within the APWG Dataset to be false positives. Therefore, CrawlPhish excludes data in the following cases:

i. *empty* websites: servers respond with no content.
ii. *error* websites: requests for URLs were denied because the phishing websites were already taken down, or used server-side cloaking which we could not bypass.
iii. *non-phishing* websites: mistakenly reported URLs, which CrawlPhish filters based on a manually curated whitelist of reputable domains.

B. Feature Extraction

Cloaked content detection. Client-side cloaking techniques used in phishing websites can be more diverse than server-side cloaking because they can not only fingerprint visitors based on configurations of browsers and systems, but may also require visitors to interact with websites. To effectively detect client-side cloaking techniques, CrawlPhish adapts J-Force: a *forced execution* framework implemented in the WebKitGTK+ browser that executes JavaScript code along all possible paths, crash-free, regardless of the possible branch conditions, event handlers, and exceptions [34]. We modified J-Force to whitelist (avoid force-executing) well-known JavaScript libraries, such as *Google Analytics* or *jQuery*, to expedite execution by ignoring the benign content changes that such libraries could introduce.

Execution time limit. We select a time limit for each invocation of forced execution by CrawlPhish to avoid failures due to long-running scrips (e.g., due to heavy branching or long-running loops). Note that this time limit is in addition to other anti-timeout features implemented in the forced execution framework, as discussed in Section IX-B.

As a starting point, we chose an execution limit of 300 seconds. We conducted an experiment by force-executing 2,000 randomly selected phishing websites in our crawled dataset to record the execution time. We found that 1.75% of phishing websites contained JavaScript code that exceeded the time limit. Execution finished as quickly as 12.56 seconds, the median execution time was 13.82 seconds, the average execution time was 29.96 seconds, and the standard deviation was 54.89 seconds. Based on this experiment, we chose a final execution limit of 195 seconds (three standard deviations above the mean) so that CrawlPhish could efficiently analyze the majority of phishing websites.

Feature extraction. To facilitate detection of (the existence of) cloaking and categorization of the corresponding cloaking type, CrawlPhish extracts both *visual* and *code structure* features from each phishing website. Each phishing website's visual features consist of the set of all web page screenshots (in our implementation, at a resolution of 2,495×1,576 pixels) captured after every possible execution path is explored by forced execution. In our dataset, each website generated 46.3 screenshots on average. CrawlPhish compares the screenshots of *each* execution path within one website against the original screenshot to detect if cloaking exists, because the presence of cloaking will result in significant visual layout changes [59]. The code structure features include web API calls, web event listeners, and ASTs, which can characterize different types of cloaking techniques and reveal how the cloaking techniques are implemented. Using forced execution, CrawlPhish can reveal and extract the web APIs and events contained in every code block, even if the code is obfuscated. CrawlPhish can then classify the cloaking types in a website using the code structure features.

Code structure features used. According to preliminary analysis which we conducted by manually inspecting cloaking techniques in a sampling of phishing websites in our dataset, different client-side cloaking techniques each have substantially different features. For example, a cloaking technique that checks mouse movement waits for an `onmousemove` event, then performs DOM substitution or redirection. However, a cloaking technique that checks screen size would first access the `screen.height` property. Therefore, as CrawlPhish executes a code block via forced execution, it records the web APIs and events that are invoked in the code block.

In addition, we found that the same semantic types of client-side cloaking techniques have many different implementations. CrawlPhish distinguishes between different implementations of each type of cloaking technique by comparing ASTs. Even though JavaScript code is often obfuscated, the AST feature is still useful because most phishing websites are deployed using phishing kits, so the corresponding websites, with the same phishing kit origin, share the same source code structure [54]. Furthermore, by computing the AST similarity, we can trace the origin of the cloaking technique by finding similar implementations earlier in phishing pages.

C. Cloaking Detection

CrawlPhish examines the *visual similarity* between force-executed screenshots and a screenshot of the website rendered in an unmodified version of WebKitGTK+ (i.e., as would be shown during a normal browser visit) to detect if cloaking exists. Because phishers implement JavaScript cloaking techniques to *evade* detection by anti-phishing systems, they

remove suspicious attributes in websites (e.g., login forms) or outright redirect to a benign website. Therefore, the visual content shown when the cloaking condition is not satisfied will differ significantly from that of the malicious page.

For example, consider a phishing website that asks visitors to click on a button in a pop-up window prior to showing the phishing content. After forced execution, two different execution paths result in two different screenshots: one as an initial benign-looking page (Figure 4a), and the other with phishing content (Figure 4b). Therefore, we consider a phishing website as *cloaked* if any of the screenshots taken during forced execution noticeably differ from the original one.

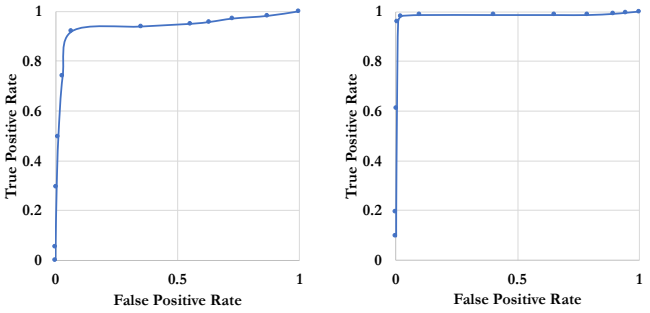
CrawlPhish can also reveal phishing content hidden behind multiple layers of cloaking. Consider a phishing website with a cloaking technique that (1) detects mouse movement and (2) checks the referrer such that the malicious content will appear only if both requirements are met. CrawlPhish will explore the execution path that shows the malicious content by force-executing it, regardless of the branching conditions. Therefore, after each screenshot is compared with the screenshot of the original page, CrawlPhish determines that a cloaking technique exists because one of the screenshots will differ.

Removal of blank pages after forced execution. Screenshots of pages rendered by force-executed paths may be blank, which can be caused by (1) negative branches from cloaking techniques (such as mouse movement detection) that require user input or (2) execution paths that take longer to finish than the execution time limit. In the latter case, CrawlPhish can mislabel a website as *cloaked* if an initial screenshot is compared to an empty page caused by unfinished execution paths. For example, phishers may trigger an infinite loop if they identify that a visit is from an anti-phishing system. In this case, CrawlPhish cannot finish forced execution and hence the screenshot remains empty. Thus, a current limitation of CrawlPhish is that it cannot detect cloaked websites with very long execution times, which we explain in Section IX. However, according to our evaluation, this situation does not happen often: only in 1.75% of the websites we considered.

Detection algorithm. To perform visual similarity checks between the screenshots, we implement the pHash algorithm [42], which compares visual similarity with robustness and good discrimination. We calculate pHash scores between the original screenshot and those captured after each path finishes execution.

$$score = pHash(S_{original}, S_i), i \in [1, 2, ..., n] \quad (1)$$

In Formula 1, S represents each screenshot and n is the number of screenshots captured from forced execution. We consider two screenshots to be similar (no cloaking) if the pHash score is less than a threshold (5.0) that we set based on preliminary testing results on 1,000 phishing websites. Differing screenshots will have a score of 5.0 or greater. Figure 3a shows the ROC curve for selecting the visual similarity threshold. We selected the threshold that provides a 92.00% true-positive rate with a 6.77% false-positive rate. We note that our evaluation in Section V shows that CrawlPhish exhibited higher detection accuracy (98.25%) with a lower false-positive rate of 1.45% than what was indicated by the threshold in the ROC curve.



(a) Visual feature threshold.

(b) Code structure feature threshold.

Fig. 3: ROC curves to select thresholds for cloaking detection and cloaking type categorization.

D. Cloaking Categorization

Once CrawlPhish detects the *presence* of cloaking on a web page, categorization of the specific *type* of cloaking allows us to measure and understand the prevalence of different high-level client-side cloaking techniques used by phishers. To facilitate this categorization, CrawlPhish maintains a *cloaking technique database* that contains the code structure features for each instance of cloaking, annotated with the corresponding cloaking semantics. Using the database, CrawlPhish can not only identify known cloaking types, but also provide detailed information about emerging cloaking techniques.

Initial database. We first obtained 1,000 cloaked phishing websites (true positives), for which we used CrawlPhish to determine the existence of client-side cloaking. Then, we manually examined the source code of the phishing websites to label the corresponding cloaking techniques. We also recorded code structure features as ground truth.

For example, we labeled one type of cloaking technique as *Mouse Detection* if the recorded code features have the `onmousemove` event and use the `window.location.href` API. Over time, as CrawlPhish executes, if the presence of cloaking is detected on a website but the code features *do not* sufficiently closely match any of the records in the database, the website is flagged for manual review such that the missing features (and, potentially, new cloaking types) can be populated. Otherwise, the website is automatically labeled with the corresponding semantic cloaking type. Within the dataset we crawled, manual effort was rarely needed after we populated the initial database. Thus, this requirement does not impede the automated operation of our framework.

Categorization algorithm. CrawlPhish employs the Hamming Distance (HD) algorithm [26] to compute the similarity of the API calls and web events. To this end, we use an array data structure with one position for each of the 4,012 types of web API calls or events as defined by the Mozilla

MDN [3, 20], which documents currently available web APIs. At each position in the array, we store the number of corresponding API calls or events as observed by CrawlPhish. We then convert this array to a fixed-length string (e.g., string[0] is the number of `ActiveXObject` in the code block and string[1] stores the amount of `Date` API calls) so that we can apply the HD algorithm. Thus, the result of the HD algorithm on a pair of strings represents the similarity of web APIs and events between two code blocks. Lower HD values indicate higher similarity.

We also leverage JSInspect [4] to find structurally similar code snippets based on the AST. This will identify code with a similar structure based on the AST node types (e.g., BlockStatement, VariableDeclaration, and ObjectExpression). We combine these approaches to overcome limitations of code similarity checkers based solely on either ASTs or API calls. Consequently, by comparing the code structure similarity of all suspicious code blocks against records in the database, all known cloaking types can be identified in one website (even if there are multiple types). If the features of a suspicious code block are not sufficiently similar to any record in the database, we will manually examine it, label the cloaking type, and then add it to the database, which is the only process that requires manual effort in the CrawlPhish framework.

Similar to the visual similarity check, we empirically set a threshold for the code similarity check based on preliminary manual analysis of 1,000 cloaked phishing websites. We consider only two categories to find a threshold: correctly labeled cloaking types and mislabeled cloaking types. Per Figure 3b, we selected a code structure threshold with a true-positive rate of 95.83% and a false-positive rate of 0.79%. When CrawlPhish compares the code structure features of a new phishing website to ones in our database, the AST similarity score must be greater than 0.74 *and* the Hamming Distance of web APIs and events must be within 34 for a new website to be marked with a known type of cloaking technique.

V. Evaluation: Detection of Cloaked Phishing Websites

In this section, we evaluate the client-side cloaking detection accuracy of CrawlPhish. In this experiment, we first randomly sampled and manually labeled 2,000 phishing websites that did not contain JavaScript cloaking techniques as well as 2,000 phishing websites with various types of client-side cloaking. We then ran CrawlPhish to detect if client-side cloaking exists. Finally, we compared the automated cloaking detection results against our manually labeled ground truth dataset to calculate the detection accuracy.

Table III shows the confusion matrix of CrawlPhish's detections. Within the 4,000 phishing websites, CrawlPhish correctly detected 1,965 phishing websites as cloaked and 1,971 as uncloaked, with a false-negative rate of 1.75% (35) and a false-positive rate of 1.45% (29). Note that unlike a general phishing detection tool that should prioritize false positives over false negatives [61], the client-side cloaking detection component in CrawlPhish does *not* critically need to do so, because the goal of our detection is to study the nature of client-side cloaking, rather than to detect a phishing attack. If CrawlPhish trades higher false negatives for lower or even zero false positives, the study might be less complete because we might miss many relevant instances of cloaking. Therefore, the detection of CrawlPhish should balance false positives with false negatives.

Crawled Phishing Websites From APWG		Analyzed			
		Cloaked		**Non-cloaked**	
Actual	**Cloaked**	TP		FN	
		1,965	98.25%	35	1.75%
	Non-cloaked	FP		TN	
		29	1.45%	1,971	98.55%

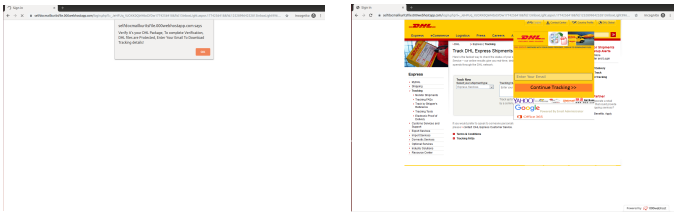
TABLE III: Accuracy of cloaking detection by CrawlPhish.

Each of the 29 false-positive cases was caused by one of two errors. The first error was due to the rending overhead of the unmodified browser which loaded the original phishing page. WebKitGTK+, the web browser we used in the CrawlPhish framework, failed to render the original websites within an allotted time limit due to a large number of CSS and JavaScript files included by the website. As a result, the original screenshot of each website was blank, but the screenshots after forced execution were not blank, so CrawlPhish mislabeled the corresponding websites as *cloaked* because the screenshots differed before and after forced execution. The second error was caused by inaccuracies in our image similarity checks. The image similarity check module erroneously distinguished between screenshots of identical pages due to slight variations in the page layout generated by the browser with and without forced execution.

In terms of the false negatives, we found that 32 out of the 35 stemmed from a long execution time of cloaked phishing websites (similar to the first reason for false positives). Forced executed screenshots are not taken if an execution path takes too long to finish execution. We used a 195-second execution time window for each execution path. However, the paths that CrawlPhish does not execute due to a timeout may contain cloaking technique implementations. Without those screenshots, CrawlPhish cannot detect the cloaking technique, so it mislabels the corresponding website as *uncloaked*.

In three rare cases, real phishing websites appeared nearly blank due to low page contrast. For example, if phishing websites have a white background with light text, CrawlPhish would not distinguish between the corresponding screenshot and a blank one. We manually examined these cases and found that CSS inclusions were missing from those websites (i.e., they could not be retrieved by our crawler).

Client-side cloaking occurrence statistics. Within our dataset of 112,005 phishing websites, CrawlPhish found that 35,067 (31.31%) phishing websites implement client-side cloaking techniques in total: 23.32% (6,024) in 2018 and 33.70% (29,043) in 2019. We note that cloaking implementations in phishing grew significantly in 2019. We hypothesize that



(a) Initial appearance. (b) Force-executed appearance.

Fig. 4: Initial and force-executed appearance of a phishing website with *Pop-up* cloaking.

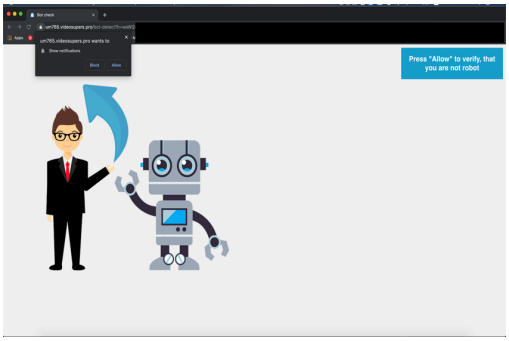


Fig. 5: A phishing website with the evolved Pop-up (Notification) cloaking technique, in which the web page directs human visitors to click on the "Allow" button by showing an arrow.

phishers are either leveraging such cloaking because it increases their profitability or because improving detection systems make advanced evasion necessary, or both.

VI. Evaluation: Cloaking Categorization

In this section, we elaborate on the eight types of client-side cloaking techniques detected by CrawlPhish (as previously introduced in Table II). We also evaluate the accuracy of CrawlPhish's semantic cloaking categorization, track trends in the deployment and evolution of different implementations of these cloaking techniques, and analyze how frequently they are used.

A. Categorization of Cloaking Types

User Interaction: *Pop-up*. With this technique, phishing content remains hidden until a button in a pop-up window is clicked. Specifically, JavaScript code listens for an `onclick` event to evade anti-phishing bots. Figure 4 shows an example of a phishing website that implements this technique. The website in Figure 4a initially shows an `alert` window to an anti-phishing bot or a real user. Thus, this phishing website seeks to evade detection by anti-phishing bots because no phishing content or typical attributes (such as a login form or logos of a legitimate organization) are found on the page. However, CrawlPhish reveals the phishing content hidden behind the popup window as shown in Figure 4b.

Figure 5 shows a more advanced version of the pop-up cloaking techniques that CrawlPhish detected. Because an `alert` window can easily be closed through common browser automation frameworks such as Selenium [28] or Katalon [5], some phishers instead use the *Web Notification API* [58]. We observed that due to technical limitations, top automation frameworks [8] do not currently support interaction with web notifications. These automated browsers opt to disable the notification window to avoid such interactions. Phishers, however, only allow visitors who actually click the "Allow" button to access the phishing content. Therefore, because the phishing website will not show any phishing content until a visitor clicks the "Allow" button in the notification window, it will evade detection. Phishers use a deceptive web page that asks visitors to click the button on the notification window, as shown in Figure 5. As an added benefit to attackers, by using a notification window, cloaked phishing websites could also directly send spam to visitors through their browsers (we do not evaluate the extent of such abuse). Through this, we show that criminals are using cutting-edge browser features to evade existing detection systems.

User Interaction: *Mouse Detection*. This cloaking type seeks to identify whether a website visitor is a person or an anti-phishing bot by waiting for mouse movement before displaying the phishing content. Specifically, the cloaking code listens for the `onmousemove`, `onmouseenter`, or `onmouseleave` events. This technique is used frequently by phishers, and accounts for 16.53% of all cloaking technique implementations in Table V, because most people have a habit of moving the mouse while a website is rendering in the browser [50].

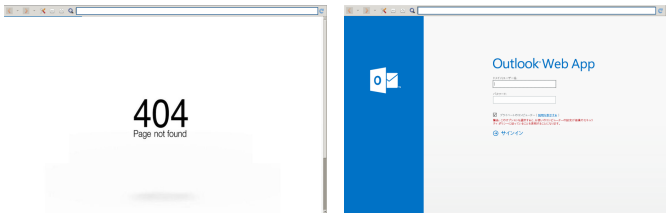
User Interaction: *Click Through*. Some phishing websites require visitors to click on a specific location on the page before displaying phishing content [60]. Simple variants of this cloaking technique require visitors to click on a button on the page and are, thus, similar to `alert` cloaking. However, more sophisticated variants display fake CAPTCHAs that closely mimic the look and feel of Google's reCAPTCHA [56]. Given the common use of reCAPTCHA by legitimate websites, phishing websites with fake CAPTCHAs make it difficult for potential victims to identify that they are fake. If anti-phishing systems cannot access phishing content because of the Click Through technique, they may fail to mark the websites as phishing.

Bot Behavior: *Timing*. Some phishing websites show phishing content only at a certain time, or deliberately make rendering slow by using the `setTimeout()` or `Date.getTime()` APIs. If phishing websites take a longer time to render than thresholds set by detection systems, such websites can evade detection. Actual visitors, however, might wait for the completion of web page rendering [19].

Bot Behavior: *Randomization*. Some phishers try to evade detection by using a non-deterministic mechanism: such phishing websites generate a random number before the page is rendered, and only show phishing content if a certain threshold is met. Anti-phishing crawlers or human inspectors may not visit the same website again if it initially shows benign content. Therefore, this technique may appear to be a "dumb" way to

Cloaking Technique		Public Dataset			APWG Dataset					Identical Groups
		Unique Groups	Top Group		Unique Groups	Top Group		Earliest Impl.	Groups Used From 2018	
Category	Type		Count	Percentage		Count	Percentage			
Fingerprinting	Cookie	43	437	15.01%	28	325	7.39%	09/2018	12	14
	Referrer	27	156	5.85%	37	92	3.92%	08/2018	21	9
	User-Agent	65	563	53.31%	33	181	12.97%	07/2018	24	20
User Interaction	Pop-up: Alert	424	249	3.26%	335	73	1.21%	06/2018	276	127
	Pop-up: Notification	29	52	4.22%	17	284	18.67%	11/2018	7	11
	Click Through	105	1,541	22.88%	51	1,275	16.45%	10/2018	13	31
	Mouse Detection	87	138	6.81%	108	500	8.63%	06/2018	47	37
Bot Behavior	Randomization	73	42	16.03%	125	58	3.57%	09/2018	62	43
	Timing	597	387	7.76%	394	416	5.99%	06/2018	303	197

TABLE IV: Overview of the number of distinct groups of cloaking code implementations in the APWG and Public Datasets.



(a) Benign page shown when cookies are disabled.

(b) Force-executed version, which reveals the login form.

Fig. 6: Appearance of a phishing website with the *Cookie* cloaking technique.

evade detection by anti-phishing systems. However, its use in the wild suggests that it may be worthwhile: we suspect that phishers who use this technique are aware of the conditions for detection by anti-phishing entities and try to trick anti-phishing bots with a non-deterministic approach to cloaking.

Fingerprinting: *Cookie*. Similar to server-side cloaking techniques, client-side cloaking techniques can also check visitors' request attributes to fingerprint them. Figure 6 illustrates a phishing website that fingerprints whether a visitor is a person or an anti-phishing bot by checking if cookies are disabled in the browser. When cookies are disabled, the phishing websites will display benign content, as shown in Figure 6a. Some anti-phishing crawlers disable cookies to avoid being bound to a single session. However, CrawlPhish detects cloaked phishing content as shown in Figure 6b. Similarly, this cloaking technique may also test if the *browser cache* is enabled [47].

Fingerprinting: *Referrer*. Phishing websites can check whether incoming traffic originates from phishers' lures or other unwanted sources. Therefore, some phishing websites display benign content to visitors with a blank *Referer* [21], which could indicate that a URL was directly typed in. Similarly, referrals from search engines or known security domains can be blocked.

Fingerprinting: *User-agent*. Some phishing websites seek to identify anti-phishing crawlers based on their user-agent strings. The `navigator.userAgent` property stores information about the browser and operating system (e.g., Mozilla/5.0 (X11; Linux x86_64)). Therefore, anti-phishing bots such as *Googlebot* can be blocked as their `userAgent` property is a known value.

Combinations of cloaking techniques. Multiple client-side cloaking techniques are occasionally used together by phishing websites, as doing so may further increase evasiveness. For example, CrawlPhish found 503 instances of *Click Through* and *Referrer* used together. Also, we found *Timing* and *Cookie* in 476 cloaked phishing websites.

B. Accuracy of Cloaking Categorization

To evaluate the accuracy of CrawlPhish's categorization of cloaking types, we selected the same 2,000 cloaked phishing websites as in Section V (this set contains all three categories of client-side cloaking techniques) and manually labeled the correct cloaking type based on their code structure features. We, then, sent these websites through the feature extraction (②) and the cloaking detection (③) phases of CrawlPhish to locate the code blocks in which each cloaking technique is implemented. CrawlPhish checked the code structure feature similarity as populated over the course of our deployment (④). As stated in Section IV-D, CrawlPhish compares the code structure features of all snippets flagged by Step ③ with the records in the database to discover all possible cloaking techniques in a given phishing website.

We found that CrawlPhish correctly categorized the cloaking type with 100% accuracy. This high accuracy stems in part from the manual inspection involved when the code structure features of the examined snippet do not match any existing records in the database, as discussed in Section IV-D. Thus, we conclude that web API calls, web events, and ASTs suffice for distinguishing between different cloaking types, even when the underlying implementations vary.

C. Grouping of Implementations

Because phishing kits directly enable the scalability of phishing attacks and are readily available through underground markets [41, 53, 54], tracking the deployment and evolution of kits can help researchers and investigators pinpoint the threat actor (i.e., a kit author or criminal group) behind a series of phishing websites and identify the prevalence phishing attacks attributable to the same author. The web page source code collected by CrawlPhish is suitable for this purpose because such source code can be obtained for virtually any phishing URL—unlike server-side code [44].

By comparing code similarity between JavaScript snippets used by cloaked phishing websites, over time, we can group

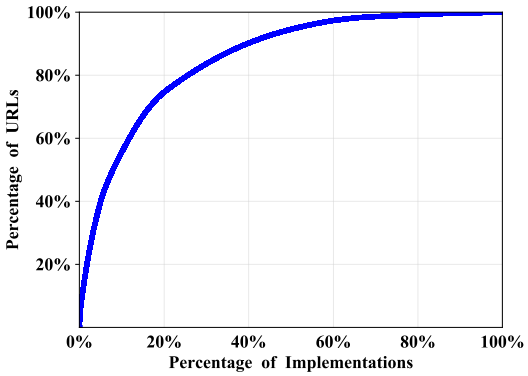


Fig. 7: CDF of implementation groups for all phishing websites in the APWG Dataset.

related cloaking technique implementations (i.e., implementations attributable to the same origin) together. Specifically, we compare the AST similarity among cloaking technique implementation code blocks to find matches using JSInspect [4] (the same technique we leveraged to check the code structure similarity). In Table IV, we provide an overview of the number of implementation groups that we found for each cloaking technique within the APWG Dataset and the Public Dataset. In addition, we compare the overlap in groups between the two datasets, and we determine the earliest date that each technique was observed.

Implementation groups in the APWG Dataset. We found that the earliest implementation of *each* cloaking type was in 2018. Also, we found that 1,128 groups account for 35,067 cloaked phishing websites detected by CrawlPhish. Figure 7 shows the cumulative distribution function (CDF) of unique implementation groups in the APWG Dataset: 20% of unique cloaking implementation groups account for 74.65% of all phishing websites. This shows that a small number of phishing kits is likely responsible for a significant proportion of sophisticated phishing websites in the wild. We discover that the Timing cloaking type has the most groups (394) among all cloaking types. Because this cloaking technique is less popular according to our findings, we suspect that prominent phishing kit developers do not deploy it, though individual criminals may still want to leverage it. Among the largest groups, we observe that one group of Click Through cloaking accounted for 16.45% (1,275) of code variants. As many as 18.67% (284) of the Notification Window occurrences were within a single group.

Implementation groups in the Public Dataset. We also compare the cloaking groups within the Public Dataset [24], which was sourced from OpenPhish [6], PhishTank [49], PhishStats [48], and other phishing URL archives. Using this additional dataset, we can validate that the APWG dataset was representative of the ecosystem and evaluate the existence of other cloaking techniques that may not have been present in the APWG dataset. Table IV shows detailed statistics on the cloaking group distributions between the two datasets. The number of groups found for each cloaking type from both datasets is similar. The Timing and Alert cloaking techniques still have the highest overall number of groups, which matches the findings from the APWG dataset. The number of groups for Click Through cloaking, however, increases from 51 to 105. We suspect that different phishers are developing more phishing kits with this cloaking technique because they realize that it can effectively evade detection by anti-phishing systems.

In addition, by comparing the AST similarity of implementation groups between the Public Dataset and the APWG Dataset, we discover that the same groups of cloaking technique types exist in both datasets. 11 out of 17 distinct groups of the Notification cloaking technique in the APWG Dataset also appear in the Public Dataset. Additionally, the Alert and Timing cloaking techniques have the most identical groups between the two datasets. This result indicates that phishing kits leveraging client-site cloaking techniques are widely used.

Evolution of cloaking groups over time. Because we crawled phishing data in both 2018 and 2019 from APWG feed, this dataset enables us to trace the origin of each cloaking type. The Timing, Alert, and Mouse Detection cloaking techniques were first used in phishing websites from June 2018 in our dataset. The (more advanced) Notification technique first appeared in November 2018. The early occurrence of these evasion methods reminds us that phishers are trying to stay one step ahead of the anti-phishing ecosystem. While researchers and anti-phishing entities were working on mitigations against server-side cloaking techniques [30, 44], those attackers had already turned their focus toward implementing client-side evasion methods. We suspect that those client-side cloaking techniques may have already been employed well before June 2018 [30, 34] (the date we started crawling).

We also observe the evolution of cloaking techniques from the perspective of obfuscation. From our crawling process, we found that the code obfuscation rate on phishing websites increased from 20.79% in 2018 to 24.04% in 2019. For example, for the Pop-up cloaking technique, the earliest variant from June 2018 was not obfuscated. Gradually, phishers started to obfuscate their cloaking technique implementations: in October 2018, they added an encoding algorithm, while the AST structure remained highly similar to unobfuscated implementations. Later, phishers started to symmetrically encrypt client-side cloaking techniques (e.g., by using AES) and included decryption keys only as request parameters. In such cases, the AST of the same cloaking technique would differ from an existing group, so we place them in a new group. However, with CrawlPhish, we still find similar web API calls, so we consider this group to be an evolution of a prior group (its *origin*). From this finding, we gain the intuition that cybercriminals are improving client-side cloaking techniques in phishing to make the latest implementations more difficult to analyze.

D. Trends in Cloaking Usage

Table V shows the prevalence of each client-side cloaking technique type that CrawlPhish detected. Note that the sum of each cloaking technique's occurrence may exceed 100% because some phishing websites implement multiple cloaking

Cloaking Technique			2018	2019	Total	Share
Category	Type		Count (%)	Count (%)	Count (%)	Count (%)
Fingerprinting	Cookie		1,295 (21.50%)	6,842 (23.56%)	8,137 (23.20%)	4,395 (12.53%)
	Referrer					2,346 (6.69%)
	User-Agent					1,396 (3.98%)
User Interaction	Pop-up	Alert	2,416 (40.11%)	17,782 (61.23%)	20,198 (57.60%)	6,027 (17.19%)
		Notification				1,521 (4.34%)
	Click Through					7,753 (22.11%)
	Mouse Detection					5,797 (16.53%)
Bot Behavior	Randomization		2,427 (40.29%)	6,141 (21.14%)	8,568 (24.43%)	1,623 (4.63%)
	Timing					6,945 (19.80%)
Total Cloaking Implementations			6,138	30,765	36,903	-

TABLE V: Cloaking technique types in the APWG Dataset, as detected by CrawlPhish.

Cloaking Technique			Total		Share	
Category	Type		Count	Percentage	Count	Percentage
Fingerprinting	Cookie		6,633	24.28%	2,912	9.87%
	Referrer				2,665	9.03%
	User-Agent				1,056	3.58%
User Interaction	Pop-up	Alert	17,634	64.55%	7,641	25.89%
		Notification			1,233	4.18%
	Click Through				6,735	22.82%
	Mouse Detection				2,025	6.86%
Bot Behavior	Randomization		5,294	19.38%	262	0.89%
	Timing				4,987	16.90%
Total Cloaking Implementations			29,561	-	-	-

TABLE VI: Cloaking technique types in the Public Dataset (September to December 2019), as detected by CrawlPhish.

techniques. In the table, the percentage under the "2018", "2019", and "Total" columns represents the share of each *category* of JavaScript cloaking technique implementation in the respective time period. The percentage under the Share column refers to the percentage of each *type* of cloaking technique in all the cloaked phishing websites we detected.

We categorize the cloaking types in phishing websites from both the APWG Dataset and the Public Dataset. As shown in Table V, the User Interaction cloaking category has the most implementations among phishing websites in the APWG Dataset. In 2018, 2,416 phishing websites (40.11%) leveraged cloaking within the User Interaction category, while in 2019, the usage ratio of User Interaction cloaking grew to 61.23%. The usage ratio of cloaking techniques in the Fingerprinting category over two years is almost the same. Within the Bot Behavior category, the usage ratio dropped significantly, from 40.29% to 21.14%. We find that phishing websites rely more on cloaking techniques in the User Interaction category than the others. We believe that this is because it is more difficult for anti-phishing crawlers to impersonate human behaviors than to bypass other types of cloaking.

Table VI demonstrates the usage of each cloaking type CrawlPhish detected from the Public Dataset. Just as we observed from the 2019 portion of the APWG Dataset, the User Interaction category was also the most frequently implemented in the Public Dataset.

Brand distribution. Among the 6,024 cloaked phishing sites in 2018, LinkedIn and PayPal were the most frequently impersonated brands, as shown in Table VII. In 2019, the distribution changed: Apple and Bank of America phishing

2018			2019		
Targeted Brand	**Count**	**Share**	**Targeted Brand**	**Count**	**Share**
LinkedIn	2,317	38.46%	Apple	6,298	21.69%
PayPal	1,104	18.33%	Bank of America	3,572	12.30%
Microsoft	646	10.72%	Facebook	2,230	7.68%
Bank of America	309	5.13%	PayPal	1,841	6.34%
Apple	153	2.54%	Microsoft	987	3.40%

TABLE VII: Top brands targeted by cloaked phishing websites in the APWG Dataset.

websites were the most prevalent. Overall, four of the top five brands in 2018 were also in the top five in 2019. Nevertheless, because of changes within the phishing landscape between the two years, our findings regarding the relative distribution of cloaking phishing websites may be skewed.

VII. Evaluation: Impact of Cloaking techniques

We have, thus far, shown that phishing websites make extensive use of client-side cloaking techniques. To demonstrate that this cloaking represents a significant threat to users, we deployed two experiments to verify that these techniques can truly evade detection by anti-phishing systems, and that they generally do not discourage victim visits—the two key factors to increasing attackers' return-on-investment.

A. *Effectiveness Against Anti-Phishing Entities*

We evaluate how effective client-side cloaking techniques are against real-world anti-phishing systems. Using a testbed for empirically measuring anti-phishing blacklists [43], we first deployed 150 carefully-controlled artificial PayPal-branded phishing websites using new and previously unseen domain names: 50 for each of the top three User Interaction cloaking types we found in the wild (Notification, Click Through with a fake CAPTCHA, and Mouse Detection). We then simultaneously reported the URLs to key anti-phishing entities across the ecosystem (Google Safe Browsing, PhishTank, Netcraft, APWG, PayPal, and US CERT [44]) to evaluate if the ecosystem can collectively detect our cloaked websites. Lastly, we monitored the detection status (i.e., blacklisting) of our websites in major web browsers (Google Chrome, Opera, and Microsoft Edge, each powered by different detection backends) over seven days.

At the conclusion of these experiments, we found that *none* of our phishing websites were blacklisted in any browser, with the exception of Click Through websites, 21 (42%) of which were blocked in Microsoft Edge a median of 3 hours after we reported them. The detection occurred because Microsoft SmartScreen classified the obfuscation in the JavaScript source code as malware, not because it was capable of bypassing the cloaking technique itself. The fact that so many of our websites remained unmitigated after a seven-day period shows that client-side evasion methods are indeed effective at evading detection by modern anti-phishing systems.

Manual inspection is used by some anti-phishing entities [23]. Recurring suspicious websites that cannot be detected by automated systems should go to manual inspection for further analysis. With specialists' inspection, any malicious

	Mouse Detection Count (%)	Click Through Count (%)	Notification Window Count (%)
Can See	879 (100.00%)	859 (97.72%)	374 (42.55%)
Cannot See	0 (0.00%)	20 (2.28%)	505 (57.45%)

TABLE VIII: Experimental results on the effect of cloaking techniques on users' ability to see phishing content.

websites therein should be labeled as phishing and be blacklisted to protect users. Our observations, however, imply that our test phishing websites may have simply been classified as benign by anti-phishing systems and never sent for manual review. We believe that this is a clear limitation of current anti-phishing mitigations. Therefore, it is important for the whole anti-phishing ecosystem to understand the nature and prevalence of client-side cloaking techniques used by sophisticated phishing websites, especially when we consider the growth of such websites [46].

B. Hampering Victim User Traffic

To verify that client-side cloaking techniques in the User Interaction category do not significantly prevent users from being exposed to phishing content on cloaked phishing websites, we conducted an IRB-approved user study through Amazon Mechanical Turk [2]. Using a free hosting provider, we generated three websites: one with each of the same three types of cloaking as considered in the previous section (Notification, Click Through with a fake CAPTCHA, and Mouse Detection). Rather than hiding phishing content behind the cloaking, however, we simply hid the text "Hello World". By default, a blank page would be shown. We then hired 1,000 workers in Amazon Mechanical Turk and requested them to report what they saw after visiting each of the three websites [1]. We choose these three cloaking techniques because they are unique to client-side (rather than server-side) cloaking implementations, and because the other techniques have been tested in a server-side context [43].

Table VIII shows the detailed experimental results. 121 of the 1,000 workers could not view our phishing websites due to a technical problem: their browsers automatically added "www" in front of the sub-domains in our URLs, which may occur in older versions of web browsers [14]. Thus, the responses of 879 workers were suitable for analysis.

For the Mouse Movement cloaking technique, 100% of the workers saw the "Hello World" text, and thus would have also seen phishing content had they visited a malicious website. For the Click Through websites, 97.72% saw the text, which shows that this cloaking technique is also effective against users. However, only 42.55% of the users saw the text on websites with the Notification Window cloaking technique. Nearly all users who did not see the text (94.94%) opted to *deny* notifications; the rest had incompatible browsers.

Although two of the cloaking techniques did not significantly prevent users from viewing the content, we found that the Notification Window cloaking technique has a negative impact on phishing success rates against potential victims. However, had these users been successfully deceived by a phishing lure (e.g., one that conveys a sense of urgency) prior to visiting the page, we believe that they would have been more likely to allow notifications [55]. Moreover, given the fact that websites with this cloaking technique were not detectable by the anti-phishing ecosystem (as we showed in Section VII), we still believe that this technique remains viable overall. In fact, the website shown in Figure 5 was still online in January 2020 even though we first observed the phishing URL in May 2019.

Consequently, we conclude that client-side cloaking techniques in the User Interaction category enable phishing websites to maintain profitability through a much longer life span, generally without discouraging victim visits, which in turn allows phishers to harm more users.

C. Responsible Disclosure

Once we established that the cloaking techniques discovered by CrawlPhish were capable of evading anti-phishing systems while remaining effective against human victims, we disclosed our findings, and the corresponding JavaScript code for each technique tested, to the major anti-phishing blacklist operators: Google, Microsoft, and Opera. All companies acknowledged receipt of our disclosure. Google followed up by requesting more information on the semantics and prevalence of the cloaking techniques, and concurred with our finding that such techniques could potentially bypass detection by current automated anti-phishing systems.

VIII. Countering Client-side Cloaking Techniques

As we have observed, phishers make extensive use of sophisticated evasion techniques in their phishing attacks. The unique feature of client-side cloaking techniques is to require visitors to interact with the website or browser, such as through a button click or mouse movement. Phishers adopt such strategies because they believe that their victims will exhibit these behaviors when visiting a website [50]. If the website is in the process of rendering and shows a blank page, most people tend to move their mouse subconsciously. Similarly, out of habit, users will click a button from a pop-up or notification window to make web page content appear. We expect that phishers' degree of sophistication will only continue to grow. Therefore, the ecosystem should ensure that existing detection and mitigation systems are capable of adapting to such evasion techniques.

To detect advanced phishing websites with client-side cloaking techniques, anti-phishing crawlers should match the behaviors that sophisticated phishing kits expect. Specifically, crawlers need to impersonate human behaviors such as mouse movement and button clicks. To examine a given website, anti-phishing systems can emulate such behaviors using automated browsers. In addition, as we observed in our analysis, the *Notification Window* technique seems to exploit the lack of support for web notifications by current automated browsers. Thus, it is important for anti-phishing systems to close this gap

and ensure that the browsers being used for detection support the same features as those used by potential victims.

Also, CrawlPhish can be directly incorporated into existing anti-phishing crawlers. With the hidden web page content revealed by CrawlPhish alongside traditional attributes such as URLs, we believe that current anti-phishing systems could identify malicious websites that would otherwise evade detection. Furthermore, by implementing CrawlPhish analysis, crawlers would be able to more accurately classify and fingerprint new variants of evasion techniques employed phishing websites, or even discover entirely new types of cloaking. Such analysis would be particularly helpful in countering phishing websites that cannot currently be classified with high confidence.

IX. Limitations

Even though CrawlPhish uncovered a diverse array of sophisticated client-side evasion techniques used in the wild, our findings should be considered alongside certain limitations.

A. CrawlPhish Deployment

Data sources. The CrawlPhish framework is not a phishing classification system. Rather, it detects and classifies *cloaking* within known phishing websites. Thus, as its primary input, CrawlPhish requires a curated feed of phishing URLs (i.e., detected by an existing anti-phishing system, whether manual or automated). However, our framework could also be adapted for use on unconfirmed phishing URLs with targeted additions to the visual similarity check of the framework [63], such that benign website screenshots could be differentiated from deceptive ones.

Data collection. Due to infrastructure limitations, we were only able to crawl live phishing websites over a total of 14 months from June to December 2018 and May to November 2019, with a 4-month gap in between. Differences in brand distribution between the two years may skew our findings with respect to the commonality of cloaking techniques. Although additional crawling would be desirable for a more thorough longitudinal evaluation, we mitigated this limitation by also evaluating CrawlPhish on a public dataset of 100,000 phishing websites from 2019, and by analyzing distinct implementations of each cloaking technique, as discussed in Section VI-C.

Phishing websites may leverage *server-side* cloaking with various degrees of sophistication [44, 46]. Although we sought to defeat simple IP and geolocation cloaking potentially implemented by the phishing websites which we crawled, other techniques may have evaded our crawler, and, thus, the corresponding phishing website client-side source code would be absent from our dataset.

Semantic cloaking categorization. When querying the CrawlPhish *cloaking technique database* to determine the type of cloaking used by a phishing website, we set fixed similarity thresholds for different classes of cloaking techniques. As a result, our approach may misclassify evasion code which combines multiple cloaking techniques, or fail to trigger manual analysis of certain novel cloaking techniques. However, as shown in our evaluation in Section VI-B, we did not observe such failures in our analysis.

B. Cloaking Detection

Execution time. Forced execution of a small percentage (1.75%) of websites in our dataset could not be completed within a reasonably short time period, and, thus, resulted in false-negative detections of cloaking. Across our deployment, we chose a 195-second idle timeout: the maximum period without a change to the execution path, after which execution is halted. This timeout allowed 98% of websites (three standard deviations above the mean) to finish, as determined by the sampling in Section IV-B. Another limitation of setting an execution time limit is that some execution paths may be omitted if the time limit is reached. A future implementation of CrawlPhish could ensure that all paths of a code snippet have finished examination by comparing the actual paths in the script to those that have been force-executed.

We found that the websites which failed to be fully executed contained long-running code within individual loops. J-Force seeks to mitigate this limitation by enforcing (by default) a cutoff of 80,000 iterations for each loop. Although this cutoff proved insufficient in the aforementioned 1.75% of cases, given the low false-negative rate, we do not consider it as a significant issue: fine-tuning the J-Force loop timeout could be used to further optimize execution times.

Nevertheless, adversaries with knowledge of our analysis technique could design code to bypass it by introducing a large number of individual loops ahead of the path which ultimately displays phishing content. To further overcome this and other types of deliberate circumvention, additional code analysis techniques, such as symbolic execution [35], could be applied in cases in which forced execution fails.

Execution environment. We force-executed phishing websites' JavaScript using the WebKitGTK+ web browser [34]. Historically, there has been evidence of malicious JavaScript that only targets a specific web browser (or engine) [32]. Thus, CrawlPhish may have failed to correctly classify code targeted at other browsers. To overcome this limitation, websites marked as *uncloaked* by the current implementation of CrawlPhish could be force-executed in additional environments.

Asynchronous content delivery. CrawlPhish does not consider cases where asynchronous web requests (i.e., AJAX) submit data about the client to the server and so that the server can determine whether phishing web page content should be sent back to the client (this equates to server-side cloaking with the prerequisite of client-side JavaScript execution, and has been previously studied [43]). Also there was no evidence in our dataset that client-side cloaking is (yet) being combined with AJAX and server-side cloaking by phishing websites. However, CrawlPhish could still be enhanced to automatically analyze the malicious use of asynchronous requests. For example, during forced execution, CrawlPhish could mutate the configurations of browser profiles before the JavaScript code sends an `XMLHttpRequest` to check for potential divergent responses. Hence, the corresponding

screenshots after mutation and forced execution would be different if cloaking techniques were dependent on AJAX, and CrawlPhish could subsequently identify the existence of such evasion.

X. Related work

Studies on phishing and cloaking techniques: Oest et al. analyzed server-side cloaking techniques within a dataset of 2,313 phishing kits and proposed a taxonomy of five different types of cloaking [44]. These authors also showed that cloaking techniques, including basic JavaScript cloaking, can effectively bypass detection by anti-phishing blacklists [43]. Based on an end-to-end analysis of large-scale phishing attacks, Oest et al. discovered that phishing websites with sophisticated evasion techniques are prevalent in the wild but the anti-phishing ecosystem has not effectively mitigated them [46]. In this work, we have presented the first in-depth analysis of client-side cloaking techniques in the context of phishing based on a dataset of 112,005 live phishing websites.

Invernizzi et al. studied server-side web cloaking techniques against search engines, and proposed mechanisms to identify and bypass such cloaking [30]. CrawlPhish leverages these methods to overcome server-side cloaking during crawling. The authors rooted their study in black markets and built a classifier to detect cloaking techniques implemented on the server side that returned different content to distinct browsing clients. This work mainly focused on the mutation of browser profiles to bypass server-side cloaking techniques to discover divergent web content. The authors found that 11.7% of search results were cloaked. The authors considered cloaking techniques used for Search Engine Optimization (SEO), advertisements, and drive-by download attacks. However, they did not investigate client-side cloaking techniques implemented in JavaScript (i.e., that execute in the browser). In contrast, we discovered diverse client-side cloaking techniques and analyzed them from the perspective of phishing attacks.

JavaScript analysis techniques: Although a number of static analysis [18, 32, 65] and dynamic analysis [34, 36] approaches have been proposed to analyze malicious JavaScript code, there has been no attempt to automatically extract JavaScript code semantics for identifying and classifying cloaking techniques. Arrow and Zozzle are static analysis methods to classify JavaScript malware based on previously discovered malicious scripts [18, 65]. Revolver tried to detect evasive JavaScript code through similarity checks against known malicious matters [32]. Rozzle is a multi-execution virtual machine to explore multiple execution paths in parallel for enhancing the efficiency of dynamic analysis so that it can be used in large-scale experiments [36]. J-Force enhanced dynamic analysis methods to find hidden malicious behaviors by force-executing JavaScript code, regardless of the conditions, to explore all possible execution paths in an automated way [34]. Hence, J-Force lends itself well to revealing content hidden behind JavaScript cloaking code.

Analysis of program semantics similar to ours has been performed within other contexts. To deal with virtualization-based obfuscation, Coogan et al. proposed a de-obfuscation approach that identifies behaviors of malicious programs based on the flow of values to system calls [15]. BEAGLE assigns semantics to malware by dynamically monitoring system and API calls that malware uses to compare versions of malicious code and quantify their differences—to observe the evolution of a series of malware [39]. Zhang et al. introduced a semantic-based static analysis approach to reveal malicious Android applications' behaviors regardless of minor implementation differences [66]. The authors leveraged an API dependency graph to determine the semantics of the program to classify malware and identify malware variants.

XI. Conclusion

Through the first in-depth analysis of the client-side JavaScript code used by phishing websites, we have uncovered a wide gamut of sophisticated evasion techniques used by attackers. In addition to categorizing such evasion techniques based on their semantics, our approach enabled us to measure the prevalence of each technique in the wild. In doing so, we observed that client-side evasion is becoming increasingly common.

Client-side JavaScript enables website developers to implement complex interactions between their websites and visitors. Thus, evasion techniques implemented in this manner pose a particular threat to the ecosystem: websites that use them can effectively discriminate between automated crawler visits and potential human victims. Unfortunately, client-side evasion techniques are difficult to analyze due to the dynamic nature of JavaScript code. CrawlPhish addresses this difficulty in a scalable manner. In addition to being able to detect and categorize client-side evasion with high accuracy, our approach can also track the origin of different implementations.

Given the rise of sophisticated phishing websites in the wild, we believe that automated analysis systems such as CrawlPhish are essential to maintaining an understanding of phishers' evolving tactics. Methodology such as ours can be incorporated by the ecosystem to more expeditiously and more reliably detect sophisticated phishing, which, in turn, can help prevent users from falling victim to these attacks through the continuous enhancement of the appropriate mitigations.

Acknowledgments

We would like to thank our shepherd, Giancarlo Pellegrino, and the anonymous reviewers for their valuable feedback. This material is based upon work supported partially by the National Science Foundation (NSF) under Grant No. CNS-1703644 and CNS-1703375, the Defense Advanced Research Projects Agency (DARPA) under Grant No. HR001118C0060 and FA875019C0003, the Institute for Information & communications Technology Promotion (IITP) grant funded by the Korean government (MSIT) (No. 2017-0-00168, Automatic Deep Malware Analysis Technology for Cyber Threat Intelligence), the NSF Center for Accelerated Real Time Analytics - NCSU, and a grant from the Center for Cybersecurity and Digital Forensics (CDF) at Arizona State University.

References

[1] "000webhost: Free web hosting," https://www.000webhost.com/migrate?static=true.

[2] "Amazon mechanical turk," https://www.mturk.com/.

[3] "Event reference," https://developer.mozilla.org/en-US/docs/Web/Events.

[4] "Jsinspect: Detect copy-pasted and structurally similar code," https://github.com/danielstjules/jsinspect.

[5] "Katalon studio," https://www.katalon.com/katalon-studio/.

[6] "OpenPhish," https://openphish.com.

[7] E. Alowaisheq, P. Wang, S. Alrwais, X. Liao, X. Wang, T. Alowaisheq, X. Mi, S. Tang, and B. Liu, "Cracking the wall of confinement: Understanding and analyzing malicious domain take-downs," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2019.

[8] B. Anderson, "Best automation testing tools for 2018 (top 10 reviews)," 2017, https://medium.com/@briananderson2209/best-automation-testing-tools-for-2018-top-10-reviews-8a4a19f664d2.

[9] APWG, "Phishing Activity Trends Report 3rd Quarter 2019," 2019, https://docs.apwg.org/reports/apwg_trends_report_q3_2019.pdf.

[10] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "Exposure: Finding malicious domains using passive dns analysis." in *Ndss*, 2011, pp. 1–17.

[11] S. Bin, W. Qiaoyan, and L. Xiaoying, "A dns based anti-phishing approach," in *2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing*, vol. 2. IEEE, 2010, pp. 262–265.

[12] A. Blum, B. Wardman, T. Solorio, and G. Warner, "Lexical feature based phishing url detection using online learning," in *Proceedings of the 3rd ACM Workshop on Artificial Intelligence and Security*. ACM, 2010, pp. 54–60.

[13] D. Canali, D. Balzarotti, and A. Francillon, "The role of web hosting providers in detecting compromised websites," in *Proceedings of the 22nd international conference on World Wide Web*. ACM, 2013, pp. 177–188.

[14] T. W. Club, "Web browser automatically adds www to url," 2016, https://www.thewindowsclub.com/browser-automatically-adds-www-to-url.

[15] K. Coogan, G. Lu, and S. Debray, "Deobfuscation of virtualization-obfuscated software: a semantics-based approach," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 275–284.

[16] M. Cova, C. Kruegel, and G. Vigna, "There is no free phish: An analysis of "free" and live phishing kits." *WOOT*, vol. 8, pp. 1–8, 2008.

[17] ——, "Detection and analysis of drive-by-download attacks and malicious javascript code," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 281–290.

[18] C. Curtsinger, B. Livshits, B. G. Zorn, and C. Seifert, "Zozzle: Fast and precise in-browser javascript malware detection." in *USENIX Security Symposium*. San Francisco, 2011, pp. 33–48.

[19] R. Dhamija, J. D. Tygar, and M. Hearst, "Why phishing works," in *Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM, 2006, pp. 581–590.

[20] M. W. Docs, "Mozilla web apis," https://developer.mozilla.org/en-US/docs/Web/API.

[21] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Rfc2616: Hypertext transfer protocol–http/1.1," 1999.

[22] Google, "Google transparency report," 2019, https://transparencyreport.google.com/safe-browsing/overview?hl=en.

[23] ——, "Manual actions report," 2020, https://support.google.com/webmasters/answer/9044175?hl=en&ref_topic=4596795.

[24] C. Guarnieri, "The Year of the Phish," 2019, https://nex.sx/blog/212/15/the-year-of-the-phish.html.

[25] Z. Guo, "World-wide cloaking phishing websites detection," 2017.

[26] R. W. Hamming, "Error detecting and error correcting codes," *The Bell system technical journal*, vol. 29, no. 2, pp. 147–160, 1950.

[27] G. Ho, A. Cidon, L. Gavish, M. Schweighauser, V. Paxson, S. Savage, G. M. Voelker, and D. Wagner, "Detecting and characterizing lateral phishing at scale," in *28th USENIX Security Symposium*, 2019, pp. 1273–1290.

[28] A. Holmes and M. Kellogg, "Automating functional tests using selenium," in *AGILE 2006 (AGILE'06)*. IEEE, 2006, pp. 6–pp.

[29] H. Huang, L. Qian, and Y. Wang, "A svm-based technique to detect phishing urls," *Information Technology Journal*, vol. 11, no. 7, pp. 921–925, 2012.

[30] L. Invernizzi, K. Thomas, A. Kapravelos, O. Comanescu, J.-M. Picod, and E. Bursztein, "Cloak of visibility: Detecting when machines browse a different web," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 743–758.

[31] T. Kachalov and zamotkin, "Javascript obfuscator," https://github.com/javascript-obfuscator/javascript-obfuscator.

[32] A. Kapravelos, Y. Shoshitaishvili, M. Cova, C. Kruegel, and G. Vigna, "Revolver: An automated approach to the detection of evasive web-based malware," in *Presented as part of the 22nd USENIX Security Symposium*, 2013, pp. 637–652.

[33] M. Khonji, A. Jones, and Y. Iraqi, "A novel phishing classification based on url features," in *2011 IEEE GCC conference and exhibition (GCC)*. IEEE, 2011, pp. 221–224.

[34] K. Kim, I. L. Kim, C. H. Kim, Y. Kwon, Y. Zheng, X. Zhang, and D. Xu, "J-force: Forced execution on javascript," in *Proceedings of the 26th international conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2017, pp. 897–906.

[35] J. C. King, "Symbolic execution and program testing," *Communications of the ACM*, vol. 19, no. 7, pp. 385–394, 1976.

[36] C. Kolbitsch, B. Livshits, B. Zorn, and C. Seifert, "Rozzle: De-cloaking internet malware," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 443–457.

[37] N. Leontiadis, T. Moore, and N. Christin, "Measuring and analyzing search-redirection attacks in the illicit online prescription drug trade." in *USENIX Security Symposium*, vol. 11, 2011.

[38] B. Liang, M. Su, W. You, W. Shi, and G. Yang, "Cracking classifiers for evasion: a case study on the google's phishing pages filter," in *Proceedings of the 25th International Conference on World Wide Web*, 2016, pp. 345–356.

[39] M. Lindorfer, A. Di Federico, F. Maggi, P. M. Comparetti, and S. Zanero, "Lines of malicious code: insights into the malicious software industry," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, pp. 349–358.

[40] "Windows defender smartscreen," 2019, https://github.com/MicrosoftDocs/windows-itpro-docs/blob/public/windows/security/threat-protection/windows-defender-smartscreen/windows-defender-smartscreen-overview.md.

[41] A. Modi, Z. Sun, A. Panwar, T. Khairnar, Z. Zhao, A. Doupé, G.-J. Ahn, and P. Black, "Towards automated threat intelligence fusion," in *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*. IEEE, 2016, pp. 408–416.

[42] X.-m. Niu and Y.-h. Jiao, "An overview of perceptual hashing," *Acta Electronica Sinica*, vol. 36, no. 7, pp. 1405–1411, 2008.

[43] A. Oest, Y. Safaei, A. Doupé, G.-J. Ahn, B. Wardman, and K. Tyers, "Phishfarm: A scalable framework for measuring the effectiveness of evasion techniques against browser phishing blacklists," in *Proceedings of the 40th IEEE Symposium on Security and Privacy (Oakland)*, Oakland, CA, May 2019, pp. 764–781.

[44] A. Oest, Y. Safaei, A. Doupé, G.-J. Ahn, B. Wardman, and G. Warner, "Inside a phisher's mind: Understanding the anti-phishing ecosystem through phishing kit analysis," in *2018 APWG Symposium on Electronic Crime Research (eCrime)*. IEEE, 2018, pp. 1–12.

[45] A. Oest, Y. Safaei, P. Zhang, B. Wardman, K. Tyers, Y. Shoshitaishvili, A. Doupé, and G.-J. Ahn, "PhishTime: Continuous longitudinal measurement of the effectiveness of anti-phishing blacklists," in *Proceedings of the 29th USENIX Security Symposium*, 2020.

[46] A. Oest, P. Zhang, B. Wardman, E. Nunes, J. Burgis, A. Zand, K. Thomas, A. Doupé, and G.-J. Ahn, "Sunrise to sunset: Analyzing the end-to-end life cycle and effectiveness of phishing attacks at scale," in *Proceedings of the 29th USENIX Security Symposium*, 2020.

[47] I. C. Paya and T. Chow, "Combining a browser cache and cookies to improve the security of token-based authentication protocols," Jul. 3 2007, US Patent 7,240,192.

[48] "PhishStats," https://phishstats.info/.

[49] "PhishTank," https://phishtank.com.

[50] T. Rotolo, "Mouse movement patterns and user frustration," 2016, https://www.trymyui.com/blog/2016/10/28/mouse-movement-patterns-and-user-frustration/.

[51] F. Shiver, "Apwg and the ecrime exchange: A member network providing collaborative threat data sharing," 2016, https://www.first.org/resources/papers/valencia2017/shiver-foy_slides.pdf.

[52] V. E. Solutions, "Data breach investigations report (dbir)," 2019.

[53] Z. Sun, C. E. Rubio-Medrano, Z. Zhao, T. Bao, A. Doupé, and G.-J. Ahn, "Understanding and predicting private interactions in underground

forums," in *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy (CODASPY)*. ACM, 2019.

[54] K. Thomas, F. Li, A. Zand, J. Barrett, J. Ranieri, L. Invernizzi, Y. Markov, O. Comanescu, V. Eranti, A. Moscicki *et al.*, "Data breaches, phishing, or malware?: Understanding the risks of stolen credentials," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. ACM, 2017, pp. 1421–1434.

[55] A. Van Der Heijden and L. Allodi, "Cognitive triaging of phishing attacks," in *28th USENIX Security Symposium*, 2019, pp. 1309–1326.

[56] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum, "re-captcha: Human-based character recognition via web security measures," *Science*, vol. 321, no. 5895, pp. 1465–1468, 2008.

[57] W3C, "Http archive (har) format," https://w3c.github.io/web-performance/specs/HAR/Overview.html.

[58] ——, "Web notifications," 2015, https://www.w3.org/TR/notifications/.

[59] D. Y. Wang, S. Savage, and G. M. Voelker, "Cloak and dagger: dynamics of web search cloaking," in *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS)*. ACM, 2011, pp. 477–490.

[60] Y.-M. Wang and M. Ma, "Detecting stealth web pages that use click-through cloaking," in *Microsoft Research Technical Report, MSR-TR*, 2006.

[61] C. Whittaker, B. Ryner, and M. Nazif, "Large-scale automatic classification of phishing pages," in *Proceedings of the 28th Network and Distributed System Security Symposium (NDSS)*, 2010.

[62] M. Wu, R. C. Miller, and G. Little, "Web wallet: preventing phishing attacks by revealing user intentions," in *Proceedings of the second symposium on Usable privacy and security*. ACM, 2006, pp. 102–113.

[63] G. Xiang, J. Hong, C. P. Rose, and L. Cranor, "Cantina+: A feature-rich machine learning framework for detecting phishing web sites," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 2, p. 21, 2011.

[64] H. Zhang, G. Liu, T. W. Chow, and W. Liu, "Textual and visual content-based anti-phishing: a bayesian approach," *IEEE Transactions on Neural Networks*, vol. 22, no. 10, pp. 1532–1546, 2011.

[65] J. Zhang, C. Seifert, J. W. Stokes, and W. Lee, "Arrow: Generating signatures to detect drive-by downloads," in *Proceedings of the 20th international conference on World wide web*. ACM, 2011, pp. 187–196.

[66] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-aware android malware classification using weighted contextual api dependency graphs," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 2014, pp. 1105–1116.

[67] Y. Zhang, J. I. Hong, and L. F. Cranor, "Cantina: a content-based approach to detecting phishing web sites," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 639–648.