School of Phish: Differentiating Phishing Pages via Browser API Usage

Anonymous authors

***Abstract*—Phishing kits, ready-to-deploy software packages for phishing websites, have proliferated the number of web phishing attacks launched daily. Last year alone, the Anti-Phishing Work Group reported over one million phishing pages. This volume presents new challenges for security researchers and analysts, as large quantities of pages employing behaviors of interest (data exfiltration, evasion, or mimicking techniques) can originate from a singular phishing kit while being deployed across different infrastructures throughout the year. At the same time, the whole ecosystem is examined page by page, and manual analysis is required to recognize static features for kit identification.**

This paper aims to aid researchers and analysts in automatically differentiating between collections of phishing pages from different underlying kits via the browser APIs executed and hierarchical clustering. We show an accuracy of 97% against a dataset of pages and kits collected from the wild. With a curated mapping of techniques to browser APIs and over 500,000 pages in which we identify 11,377 clusters, we explore what techniques are universal, widespread across kits, or kit-specific. Prior work has noted an increased complexity of client-side JavaScript included in phishing pages; overall, our methods and findings show that browser API usage can leverage this against adversaries to differentiate phishing pages originating from different kits and better understand the breakdown of behaviors in the ecosystem.

I. Introduction (1 page)

Web-based phishing attacks, where a web page, through mimicking or urgency, tricks the user into submitting personal information to an attack, have been increasing for the last 5 years[APWG]. While varying in delivery vectors, sent through email, SMS, or QR codes, they remain a successful attack for compromising individual accounts and enterprises. In 2021, the Cybersecurity and Infrastructure Security Agency (CISA) warned of phishing campaigns targeting NGOs and government workers which smuggled the HTML and Javascript of the phishing page through an attachment[https://www.microsoft.com/en-us/security/blog/2021/05/27/new-sophisticated-email-based-attack-from-nobelium/, https://www.cisa.gov/news-events/cybersecurity-advisories/aa21-148a]. Once an actor steals credentials, they sell them on illicit markets or leverage them to leak more data from the target, for example in 2024, joined with the Multi-State Information Sharing and Analysis Center, CISA reported that compromised former employee credentials were being used to access internal networks within the US government[https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-046a].

Client-side javascript is a keystone in this profitable and effective family of attacks. This highly dynamic language, which has access to privileged functionality through browser APIs, enabled the attacks to obfuscate the true purpose of the web page (HTML/CSS and the javascript code itself) and effectively identify settings when a victim or a potential analysis framework is viewing the web page. [https://developer.mozilla.org/en-US/docs/Web/JavaScript, CrawlPhish, Spartacus]. The javascript logic can range from a simple user-agent-based redirection to an AES-encrypted script that dynamically decrypts itself, identifying the browser through a series of API calls extracting information about the host's browser capabilities. The ability to distill information about the user via the browser's capabilities is referred to as browser fingerprinting. Prior work has studied it in both phishing and web-based advertising settings.

The Anti-Phishing Work Group (APWG) has reported over 1 million phishing attacks yearly since 2020. With the ecosystem full of pages, with Javascript reaching browser APIs to mimic the target page, cloak away from analysts, and exfiltrate data, we propose using browser API traces from these pages to study them at a layer of abstraction. kit We have crawled five phishing feeds for 401 days to answer the following research questions.

RQ1. **How indicative is a page's browser API trace of the underlying phishing kit? What differences do we see between the kits?** Having collected 4,448 pages from 526 distinct kits, we find that clustering based on the distance of browser APIs used is enough to identify if two pages share a kit behind the scenes, achieving an accuracy of 92% based on Fowlkes–Mallows index.

RQ2. **What does the clustering of all phishing pages collected look like?** By computing the density based cluster of these pages for a 4-week window, rolling it by 2 weeks, and merging these clusters by threshold of common pages, we get 27,833 total cluster for the span of 401 days. These clusters are very well formed with a silhouette score of 0.83 ($\sigma = 0.04$).

RQ3. **How are browser APIs being used in phishing? How are different techniques spread across clusters?** We find that 1.2 million pages (83%) of pages we observe execute JavaScript in a first-party context. We find no more then 15% use an obfuscation related browser API. We check API usage of techniques established by prior work and find a decrease in user-interactive permission pop-ups. We do see that 1 in 3 pages calling out to a basic fingerprinting property (userAgent, cookie, or HTTP referrer). We observer that 1 in 4 pages makes a network request upon loading, some reaching out to IP identification URLs which allow free look-ups for the requesting machine, in this case that being the victims

browser.

The rest of the paper will breakdown the necessary background regarding the phishing ecosystem and adversarial techniques, we will go over relevant work in studying the ecosystem in Section-III. We will describe the methods we used to study the ecosystem in Section-IV and go over what we found in Section-V.

II. Background (1.5-2 pages)

In this section we provide a helpful background on current developments in phishing as a phenomenon and adverserial javascript techniques, as well as an overview of the parties involved in the phishing ecosyste.

A. Phishing as a phenomenon

Phishing is a form of social engineering where an adversary pretends to be a trusted entity to steal a user's credentials or gain access to a specific machine, network, or account to which the user has access. While delivery mechanisms vary, the most common way of phishing inevitably leads to a web page that requests some personal information (usually credentials) from the user.

While a basic phishing site is relatively simple to deploy and send through social media or email, the ecosystem has proven to be an ever-changing landscape. Workgroups tracking phishing saw an uptick in phishing domains in 202X, with a shift in what sectors are targeted throughout every year. In Q3 of 2020, the APWG reported SAAS and webmail services as the most targeted sectors, with 31% of all phishing attacks targeting them. In Q3 of 2021 it was the financial sector and in 2024 it is Social media[1]. In the 3rd quarter of 2024, the Anti-Phishing Working Group (APWG) reported 900,000 phishing attacks. Besides detection, to be able to identify trends in the ecosystem, security researchers need to be able to group a large number of pages through a higher level of abstraction.

As more organizations and academics turn to studying phishing websites and develop methods of trying to identify them, anti-analysis and anti-bot detection techniques have emerged in the ecosystem. These are no different from anti-debugging techniques found in malware; the end goal is to make detection and analysis of the page as tricky as possible. These techniques can be broadly classified into server-side and client-side techniques. Server-side techniques are stealthier; however, they rely on limited information. These can be studied by getting your hands on a zip bundle to set up the page (sometimes called a phishing kit). Most server-side techniques rely on pre-compiled denylists or allowlists of IP addresses, user agents, or referrers (the page from which the link is visited as identified by an HTTP header). The cloaking outcome can also vary; the page can redirect to a benign page, a long-dead phishing page, or send you off to an affiliate marketing page. Some pages may be simple checkpoints and choose to forward you to the actual phishing page should they decide you are not an analyst or a bot.

Client-side techniques, while allowing for much richer evasive behavior, are more detectable via instrumented tools. With the rise of fingerprinting on the web, driven by tracking scripts of advertisers and analytics companies, phishing pages turned to using browser fingerprinting APIs to identify real users and exfiltrating the fingerprinting to associate with the credentials harvested. Phishing pages utilize Browser APIs to trigger permission pop-ups to identify headless browsers (without a user interface usually used for crawling) and automated crawlers that can not interact with the full browser UI. Prior work has also identified captchas and click-through pages as a form of client-side cloaking, which requires more sophisticated crawlers to combat.

Even URL features of a phishing link contain techniques that have evolved to respond to anti-phishing research. Phishing pages frequently use URL shorteners (public or private) to obfuscate the final destination, landing pages requiring a user to follow hyperlinks to the actual page, and free web hostings with trustworthy TLDs.

B. Phishing as an ecosystem

As an ecosystem, phishing attacks have two major branches: cybercrime as a service and credential sales. While you could develop a phishing page, set up hosting for it, adapt the latest cloaking techniques to your page, deliver the page through SMS, Email, or social media, and leverage the accounts for the financial game, the ecosystem as a whole has compartmentalized these into different segments. Phishing kits (bundles of software providing ready-to-deploy phishing pages) rose in popularity in illicit markets, varying in value, sophistication, and features. One stage higher from just selling kits is Phishing-as-a-service providers, which will do the majority of the technical challenges. Both of these enable massive phishing attacks with lower technical barriers to entry, so naturally, they attracted the attention of researchers and analysis studies. Prior work has shown that phishing kits are not above stealing from their costumers [13], [31], adapt and borrow features from others [20], and sometimes are tied to specific actors [5].

The credential sales part of the ecosystem has also adapted to modern MFA/2FA practices. With prior work showing that a browser fingerprint is sometimes enough to trick their protections[27], an actual cost is associated with executing javascript on the phishing page[45], as accounts with a browser fingerprint go for a higher value on elicit markets. While phishing pages can risk storing the credentials on servers that can be taken down, some have shown that they should be sent to chat channels for ingestion.

C. Adversarial JavaScript

The dynamic nature of javascript enables a variety of techniques for concealing from analysis and detection. Javascript obfuscation can transform a known malicious sample into an undetectable one. WebPack enables bundling benign and malicious scripts and wrangling them to make static analysis harder. The other side of obfuscation is evasions; while not nearly there to make code comprehension (via human or machine) harder, malicious actors have deployed time bombs,

offloading parts of the malicious script to be read from the DOM or via a network request, and dynamic code generation to evade revealing the entire malicious behavior and thus detection.[41]

The research community has tried tacking these techniques by developing deobfuscators, employing machine learning techniques from static features to identify similarities between known malicious and unknown samples, and developing force-execution engines, executing parts of the code that may trigger the malicious behavior or the next stage of an evasion.

III. Related Work

A. Phishing

Phishing detection: There is a wealth of research on phishing detection as the tug-of-war between adversaries and security professionals continues. Recently, Liu *et al.* and Abdelnabi *et al.* have deployed vison-based techniques to detect phishing pages [30], [6], [29]. [30] also presents over 6,000 phishing kits analyzed as part of the work. With adversarial attacks ensuring that the page looks different to crawlers and analysis, some have turned to extracting features from the URLs themselves, more recently via LLMs in [11], [22] and before via statistical models and machine learning in [48], [24], [43], [54]

Studying and combating adversarial techniques: Divakaran *et al.* in [14] reaffirms the need to keep up with the latest adversarial techniques to build detection systems for phishing better. Prominent work in this area includes [56] by Zhang *et al.*, which uncovered and categorized many novel client-side techniques by force executing phishing pages to trigger the cloaking behavior. Acharya *et al.* in [7] uncovered that phishing pages can successfully evade blocklists by knowing how to identify their crawlers, and Oest *et al.* in [36] demonstrated that cloaking from non-mobile based devices as a phishing page can ensure that your page goes unmarked by the blocklists for more than 48 hours.

Kondracki *et al.* in [23] uncovered a massive blindspot of the phishing detection ecosystem that was Man-in-the-middle phishing kits. Kits that would transparently forward your connection to the target page, mimicking brand logos on pages like Outlook without any configuration. Fortunately, the authors addressed the blindspot by demonstrating that these proxies remain fingerprintable using TLS fingerprinting. [53] proposed a similar attack, however, one that used javascript and NoVNC to trick the user into signing into their account through a VNC session in their browser.

With adversaries becoming creative with their evasions and obfuscation techniques, some novel defenses have also opted to think outside the box. Zhang *et al.* in [57] proposed a phishing defense solution that leverages the high likelihood of a phishing page cloaking away from a crawler to the defender's advantage. They demonstrated that a web browser configured to look like a crawler will trigger a cloaking response from phishing pages, ensuring that your users never see the page while maintaining compatibility with all of Alexa's Top One Million websites. Meanwhile, to combat phishing pages using captcha's [51] utilized vision-based models.

To better understand why, other than cloaking, phishing pages may choose to fingerprint, Lin *et al.* in [28] showed that browser fingerprints could be successfully used to bypass multi-factor authentication, a system meant to be a last line of defense against stolen credentials, for 10 out of 16 websites that provide popular services.

Phishing kits: Much can be studied about the phishing ecosystem via phishing kits. Cova *et al.* in [13] uncovered that most "free" phishing kits contain a backdoor, effectively serving as a way to offload the deployment of a campaign to a 3rd party while siphoning off their stolen credentials.

Similar to our goals, [10] using a dataset of phishing kits gathered through **Kitphishr** and a collection of features extracted from the HTML DOM to classify websites into their matching kit. They achieved an F1 score 91 when classifying 2,000 pages (1,141 benign and 859) from features extracted from their kit. However, their multi-class classifier for identifying the kit only achieved an F1 score of 39. Merlo *et al.* in [32] further expanded on our understanding of phishing kit lineages by looking at over 20,000 phishing kits and identifying, via token similarity, most of them as clones of one another or previously encountered kits. Prominent work in extending our understanding of phishing attacks includes Han *et al.* in [19], where they monitor the deployment of phishing kits by adversaries that compromise vulnerable web servers by hosting a well-sandboxed honeypot. They collected 643 phishing kits and established that kits take minutes to install and test and can remain undetected for weeks. Using these kits, they were also able to identify evasion techniques used by these kits, like path randomization per visit, which back then was enough to bypass Google SafeBrowsing.

In [37], Oest *et al.* manually analyzed phishing kits to establish the taxonomy for server-side Cloaking, and in [9], Bijmans *et al.*, after collecting phishing kits by watching TLS transparency logs to identify Dutch brank phishing domains, manually creates a fingerprint from static features to analyze their prevalence in the wild.

More recently, Lee *et al.* in [25] provides a server-side script (PHP) level analysis of phishing kits, finding that dynamic generated URLs are still standard in the ecosystem and seeing seasonality in the kits they were able to obtain.

Extending the understanding of the phishing ecosystem:

Like our methods, Rola *et al.* in [45] deploy a modified Chromium browser to study a pre-selected browser API list on phishing websites for first and third-party scripts. They find that most of the most visited phishing pages (identified via browser telemetry data) deploy fingerprinting scripts, sometimes varying from the ones of the original brand they portray. At the same time, they accessed this at a script level, which reinforced our finding that phishing pages vary vastly from their original page.

Oest *et al.* in [38] demonstrates the full lifecycle of a phishing campaign by employing the fact that phishing pages often copy assets from the target domain and refer the victim back to the original page afterward. By collaborating with a significant financial institution, they developed a framework for leveraging this data to track a phishing page from its deployment to

blocklists flagging the page as phishing. [38] observed all techniques highlighted by prior work, Cloaking, user-specific URL generation, Man-in-the-middle proxies, and short-lived bursty attacks. Expanding our understanding of the victim experience on a phishing website, Subramani *et al.* in [50] developed a crawler.

B. Dynamic analysis of web pages

This work shares methodology to dynamic analysis enabled by web measurement frameworks like OpenWPM[15] and VisibleV8[21]. Su *et al.* used VisibleV8 traces and taint analysis in [49] to discover emerging fingerprinting techniques. Sarker *et al.* used VisibleV8 to create an oracle for detecting obfuscation in [46]. Such an oracle was made possible by the observation that VisibleV8 marks the execution of an API at a given source line, At the same time, obfuscation techniques ensure that the API is not textually available there. And Pantelaios *et al.* in [41] used a combination of VisisbleV8 and force execution modifications to the chromium engine to identify and defeat javascript evasion techniques while also leveraging API traces and clustering to identify previously unlabeled malicious extensions.

IV. Methodology (1 page)

This paper aims to study the evolution of browser API usage in phishing pages and leverage execution traces of these pages to identify which ones share an underlying kit. To accomplish this, we require browser API execution trances from phishing pages over a long period and a subset of pages where we know the underlying phishing kit. In the following section, we describe the experimental setup for gathering this data, the steps we took to aggregate and enrich the execution traces, how we identified anomalies and trends, and finally, the steps we took for clustering the data and evaluating them as an analog for phishing kits. A full overview of our crawling and analysis pipeline can be seen in 1.

A. Data Gathering

URL feeds: We gathered phishing pages from a diverse set of phishing feeds, monitoring OpenPhish[40], PhishTank[12], URLScan[47], SMS Gateways[35], PhishDB[33], and APWG[8], based on the availability of the feed and the level access we had at the time. Every hour, we checked these feeds for new urls (limited to the last 48 hours) and submitted them into two different crawlers: VisibleV8 and KitPhished.

VisibleV8: To get execution traces for every script loaded when visiting the page, we used an automated chromium-based crawler with VisibleV8 patches applied. The browser is being automated to visit the page and take screenshots with puppeteer[18], an NPM package by Google to help in UI/UX testing and browser automation. The patched chromium crawler uses puppeteer-stealth, a set of configurations to help mask the headless chrome and puppeteer itself from detection tools[39]. We initiated the crawls from a university network designated for research purposes and used Catapult[17], a man-in-the-middle proxy, to capture the entire HTTP archive for replayability. The crawler stays on the page for 45 seconds before taking a screenshot to allow scripts to load and start executing; this is consistent with prior work[21], [15].

KitPhisher: While not guaranteed, some malicious actors leave the zip files of the kits used in a discoverable folder on the same server that hosts the website (for example, the Apache document root). KitPhisher[2] is a go-based URL fuzzer that attempts to identify any left-over zip files on the server. Prior work[29], [37] establishes it as an excellent way to collect and analyze phishing kits. If it is successful, it will download the zip file and make a note of the domain from which kitphishr acquired it.

Brand's Original page: OpenPhish and APWG's eCrime Exchange report the brand that a phishing page targets. We selected 10 out of the top 100 brands targeted and collected VisisbleV8 logs for their home pages and, when applicable, their login pages. This is to assess how similar phishing pages are to their target pages. Lin *et al.* in [27] identified that phishing pages could effectively use browser fingerprints to bypass multi-factor authentication, and [45] found that phishing pages deploy a plethora of 3rd party fingerprinting scripts, which differ from the scripts of the original page, what is the simularity between browser API usage between the target brand's page, and the phishing counterpart.

B. Data enrichment and De-duplication

1) Enriching browser API executions: **Browser APIs and property accesses**: VisibleV8 has a default log-postprocessor set. These programs take the raw lags generated by the patched Chromium browser and convert it into an organized database, identifying duplicate scripts via sha3 hash, clearly marking the origin of each script that loaded and isolating which javascript API calls are browser API calls defined in the WebIDL file[1] generated during the while building the patched chromium. For our analysis, we use a tuple of the original page's URL, the script's URL, and a set (unordered) of APIs executed by this script.

First party API sets: To cut through the noise of cloaked pages and include third-party scripts like Google Analytics, known to be present in phishing pages[26], we isolate API sets executed by first-party scripts (from now on called first-party API sets). To do this, we establish the root domain as the domain submitted to the feeds and the origin as the domain from which the script is loaded. We consider a script first-party only if loaded from the domain we acquired from our feeds (root domain).

Browser APIs of interest: Even focusing on APIs that WebIDL marks, the total number of browser APIs exceeds 15,000 (depending in how you choose to categories inherited functions). We leverage Mozila Developer's network 150 WebAPIs categories when aggregating all the APIs used. MDN will mark which specification's (for example Web Crypto, Fetch, or HTML DOM) modify the implementation of a given method call, constructor, or property access. This can result in

[1] https://developer.mozilla.org/en-US/docs/Glossary/WebIDL

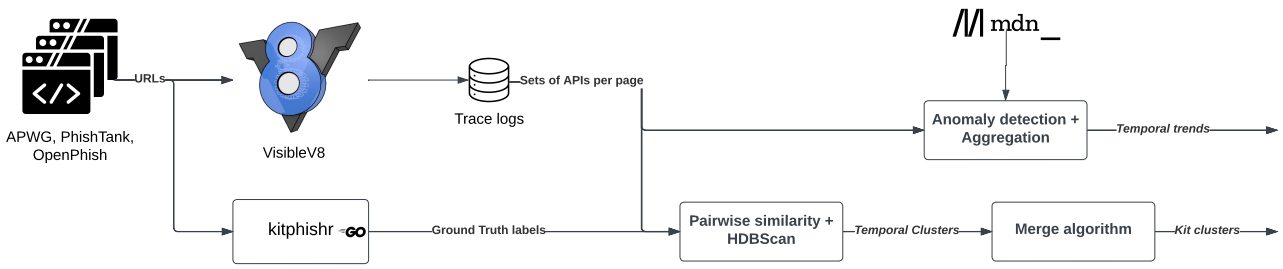


Fig. 1: Crawling and Data enrichment infrastructure

'Window.fetch' falling both under 'HTML DOM', 'Fetch API', 'Streams API', and 'Attribution Reporting API'. To resolve this, we take in the ≈ 500 APIs that have multiple mappings and resolve them into a single category based on our prior knowlage of browser functionality.

However, we also look at prior work by Su *et al.* and Zhang *et al.* for categories of Browser APIs for fingerprinting and general cloaking, respectively. We use MDN's documentation to identify a list of 22 APIs that trigger a permission pop-up or validate permission. We should note that while *Permissions.query* is the only way to verify permissions, different WebAPIs like USB, clipboard, Geolocation, and Notification have their different APIs for requesting the pop-up

De-duplicating kit files: We crawl the URLs with KitPhisher to establish a ground truth dataset with URLs originating from the same kit. For zip files extracted via KitPhisher that have password-based encryption enabled, we use the SHA256 hash of the zip files to identify which domains yielded the same kit. For the remaining zip files, we further de-duplicate them into Kit-Families by looking at them as a set of SHA256 of source files[2]. If the Jaccard index-based similarity ($JI(A,B) = \frac{|A \cap B|}{|A \cup B|}$) of these sets is equal to or greater than 90%, we consider the two kits to be from the same family, and thus group all domains from both kits into the same group.

Adjusting for Cloudflare: We find a high number of anomalies originate from cloudflare scripts on the same domain as the phishing pages. Since cloudflare scripts load from a url with a **'cdn-cgi'** in the url, for most of our analysis we do not consider scripts loaded from that endpoint. We however, we do discuss the behaviors we were able to see from these scripts in Section-V.

C. Identifying trends

Identifying anomalies: We use rapture[52] to detect change points in the time series of API calls. We use a linearly penalized segmentation algorithm (PELT), which combines a penalty parameter β and a cost function to quickly partition a time series based on points where it detects the series starts to change. For the cost function, we use **L1Cost**, which uses a deviation in the mean, and for the penalty parameter, we use $\beta = 2\ln(n) = 12$ where n is the number of days in the timeseries. This aligns with the recommendation of Killick*et al.* in the paper introducing PELT to minimize the Bayesian information criterion. We normalize by our daily URLs ingested and look at a time series of the percentage of daily URLs that executed the given URL. To clear up the results from rapture, we look at the change in the mean between segments. Any chance being less than 1%, we ignore and merge with the prior segment.

[2]Source files identified via libmagic

TABLE I: Manual criteria to identify cloaking behavior established by Zhang *et al.*

Cloaking Category	Required API calls
User Interaction	Any permission requesting API call FilePicker API calls HTMLElement event listeners
Fingerprinting	HTMLDocument.cookie HTMLDocument.referrer Navigator.userAgent
Bot Behavior	Crypto.getRandomBytes[3] Window.setTimeout + Performance.now

Identifying different traces from the same script: VisibleV8 will de-duplicate the same script loaded by multiple pages via the sha3 hash of the script. To establish if scripts have non-deterministic behavior, we pull the set of APIs executed by the script and compare the set of APIs executed by the same script on different pages. We use the Jaccard index to establish the similarity between the sets of APIs.

D. Identifying kits

We hypothesize that similarity in browser API execution means that the pages originate from the same phishing kit. We use the Jaccard index on the API sets that first-party scripts execute to establish this similarity. We use HDBScan, with a minimum cluster size of 10, to cluster the pages based on the Jaccard index. We then use the ground truth labels from KitPhisher to evaluate the clustering. When ground truth is available, we evaluate the clustering using the Fowlkes-Mallows Index (geometric mean between precision and recall)[16] and V-measure.

V-measure is a harmonic mean between completeness (all members in a cluster are from the right class) and homogeneity (all members in a cluster are from the same class). V-measure allows tuning a ratio β that prioritizes the score towards one vs the other. More importantly, looking at completeness and homogeneity scores lets you see how your clustering approach is getting things wrong[44]. We use sklearn's measure module to calculate all cluster evaluation metrics.[42].

For the much larger set of pages for which we do not have kit labels, we use silhouette score[4], a metric relative to how well packed and separated the clusters are, to evaluate the clustering when we do not have ground truth. As this would require clustering a total of more than 800,000 pages, we instead cluster a rolling window of 4 weeks, that we slide by 2 weeks. We then merge these temporal clusters if two clusters from different timezones have at least five common pages (same as our minimum parameter for HDBSCAN).

V. Results (3-4 pages)

In this section, we will evaluate our clustering approach against the ground truth, inspect the distribution of ground truth pages in the final clusters, and report temporal patterns and the commonality of phishing behaviors across those clusters. In total, we crawled for 439 days collecting browser traces from 1,328,917 pages, out of which only 533,514 qualified for clustering by executing at least 8 APIs. Further more, 99,464 pages were clustered as noise by HDBSCAN or formed a cluster where all the pages had no common APIs.

A. Characteristics of clusters

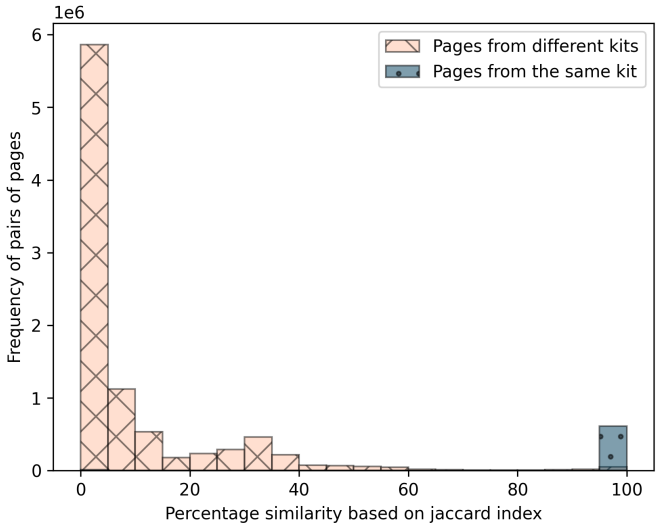


Fig. 2: Histogram of the Jaccard index based simularity between any pair of pages belonging to the same kit, versus from different kits

Finding 1: *Pages from different kits employ vastly different APIs*. Figure-5 shows the distribution of Jecard-based similarity between pages from the same kit versus different kits. We observe that pages from different kits, even ones with similar themes, rarely exceed 60% similarity, while most pages with the same kit have around 90-100% similarity. We leverage this in our clustering by treating $1 - JI(A, B)$ as a distance kernel of HDBSCAN.

Finding 2: *Browser API usage unique identify phishing kits apart from eachother*. Clustering pages from 4,448 pages across 521 kits, yeild 349 clusters. Evaluating these clusters against the ground truth labels for each page, we find that our clusters have an FMI-based accuracy of 0.93. The clusters have a V-Score of 0.86, which increases with a higher β, meaning the clusters tend to combine two kits into one instead of splitting pages from the same kit across multiple clusters.

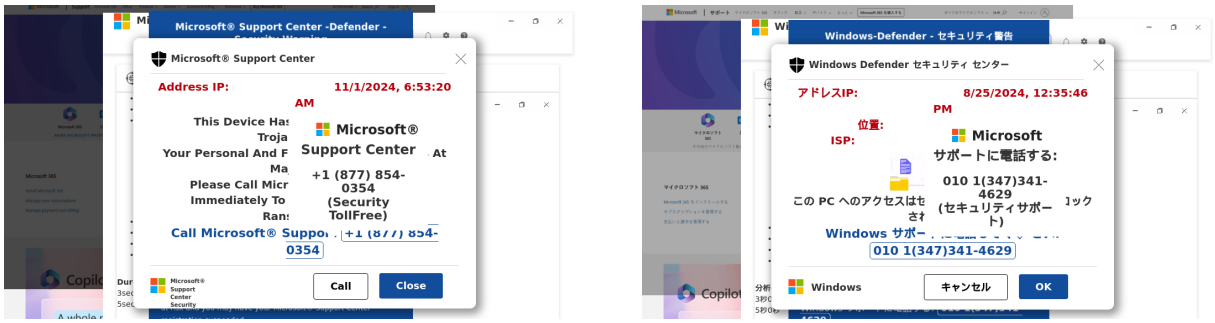
Pages with ground truth labels for originating kits remain appropriately sorted out in the clustering of the 434,050 pages, as out of the 3,505 pages clustered within the larger clusters, we maintain an FMI of 0.948 and a V-Score of 0.889, respectively. Manually inspecting the clusters, we observe that these clusters unique pages across deployment types (AWS, Cloudflare, DigitalOcean, etc.) and languages. For example, Cluster-e325887b comprises 487 pages across 5 unique e2TLDs and contains pages engaging in voice-based phishing attacks (tech support scams) across Japanese, English, and German, varying phone numbers and errors in each, shown in Figure-3. With over 400 APIs in common, it is clear that these pages' usage of keyboard intercepting APIs, Audio APIs, and Network APIs for IP intelligence caused the cluster to be formed.

Finding 3: *DOM APIs and property accesses play an essential role in identifying the underlying kit*. With 17,753 browser APIs, feature reduction becomes an obvious goal. However, removing DOM-related APIs, or property reads, out of consideration drastically reduces the number of pages we can consider for ground truth evaluation, not increasing our overall accuracy. Evaluated our same methodology described in Section-IV with all HTML-DOM, SVG, and CSS APIs removed and observed FMI-based accuracy of 0.93 and V-Score of 0.86. When all property reads were removed (often used in finterprinting[49]), we saw FMI and V-Score of 0.93 and 0.85, respectively. In both cases, we can cluster fewer pages and thus identify fewer kits.

Finding 4: *Majority of clusters map to a single brand*. 80% of clusters contain urls only marked by a single target brand by our threat intel sources[5]. 534 clusters (15%) had two brand labels, however the most popular combination of these were "Meta/Facebook", "National Police Agency JAPAN/Facebook", and "Facebook/Instigram", keeping the parent organization of the target the same in majority of the cases. Manual examination of clusters with "National Police Agency JAPAN/Facebook" brand labels revealed shopping pages in Japanese to be marked with that label in correctly from our data feeds. The cluster with the most diverse set of brand labels had 14 unique brand labels, which was a cluster with 12,467 pages with simple sign-

[4]While this metric is biased against non-convex clusters, based on our results, we do not see it necessary to switch to a density-specific cluster metric

[5]We did not include clusters in this count that had no brand labeled urls in them



(a) Page variant in English

(b) Page variant in Japanese

Fig. 3: Cropped screenshots from Cluster-e325887b, IP addresses and location redacted to ensure anonymity of the authors.

in pages that exfiltrated information using client-side registered event listeners to exfiltrate data using client-side javascript.

B. Temporal patterns of clusters

Finding 5: *Majority of phishing pages that execute originate from the same 50 clusters*. 40.7% pages of pages that executed javascript in a first-party context (533,514) are sorted into one of the 50 top clusters. The 10 clusters by page count are 20.8% of the total pages alone. We provide a breakdown of these clusters along with manual labeling of what campaigns they corespond to in Figure-4. We note that one of the clusters, we simplly labeled dynamically generated, as this turned out to be a noisy cluster of simple pages that dynamically generate part of their page, without any sophisicated client-side behavior. The E-comerse cluster shown in, Figure-4 has a significant seasonalality as lags 7, 14, and 21, meaning the appearence of the clusters on the feeds happens every week. However, further investigation showed that majority of seasonal clusters are simular e-comerse phishing clusters, with vastly different dynamic behavior, letting us to conclude that the seasonalality in majority of the clusters is due to regular reporting by threat-intelligence sources to our feeds, and not seasonal deployments of kits.

Finding 6: *72% of the clusters (144,695), and by extension kits, are only seen for a single month by phishing feeds*. 15% of the clusters (14,555 pages) we observe are only seen for one day, along which were pages deployed on blogpost, brandless bank pages, and phishing page impercenating the goverment of Korea. Meanwhile, 1,282 clusters (11%) have lifetimes longer then 100 days.

Finding 7: *Clusters re-emerge after some downtime*. Some clusters that have a lifetime greater then 100 days, still only deploy a few pages (less then 1 page every 10 days). We use this as a heuristic to identify 251 clusters 're-emerge' through our observation period.

C. Phishing Techniques across clusters

Finding 8: *UI interactivity and fingerprinting are a near universal behavior across clusters*. Multi-stage phishing pages are very well documented in prior work, and we find that majority of clusters (91%) register a click event listener using JavaScript. Though this could be as simple as submitting credentials using JavaScript, this highlights the need for researchers to augment their crawlers in the future to extract better and more complete execution traces from websites. As mentioned in Section-IV we split fingerprinting into two categories, basic and advance. Basic fingerprinting, which follows the list of APIs identified by Zhang*et al.* in [55] was present in 80% of the clusters (over 300,000 pages), and Advance fingerprinting (measured by at least 5 APIs idenfied by Su*et al.* in [49]) show up in 70% of the clusters. Together, 85% of clusters (9572 clusters, 313,212 pages) exhibit some kind of fingerprinting.

TABLE II: Breakdown of the obfuscation techniques observed in our dataset

Obfuscation techniques	Clusters	Pages
Window.atob	61,125	1,455
eval	14,561	982
Textdecoder.decode	11,113	534
SubtleCrypto.decrypt	1,185	36

Finding 9: *Fingerprint exfiltration, obfuscation, and bot detection are wide-spread phishing clusters*. While fingerprinting is near universal, we find that smaller fraction of the clusters employ obfuscation, fingerprint exfiltration, and timing for bot detection. We present the breakdown of all these techniques in Table-II. We find that 22% of clusters call out use Performance.now inconjectuion with setTimeout, enabling delta time measurement for bot detection. 46% of pages call to at least 5 advance fingerprinting APIs, followed by an exfiltration related API.

31% of clusters (2,395) employ some form of obfuscation. A full breakdown of different obfuscation forms is in Table-II and as we can see eval and Base64 encoding were the most popular way of obfuscaiton. Despite best recommendations[4] to web developers, javascript's eval function remains a favorite for obfuscation and evasions [41]. Sometimes, a script is executed via 'eval()', which evaluates yet another script itself; we measure this phenomenon as a level in **eval-depth**. We find that 48 clusters have pages that go to eval-depth 3, however, this seems to be a side-effect of embeding the phishing pages (mainly ones targetting facebook) in a blogpost page.

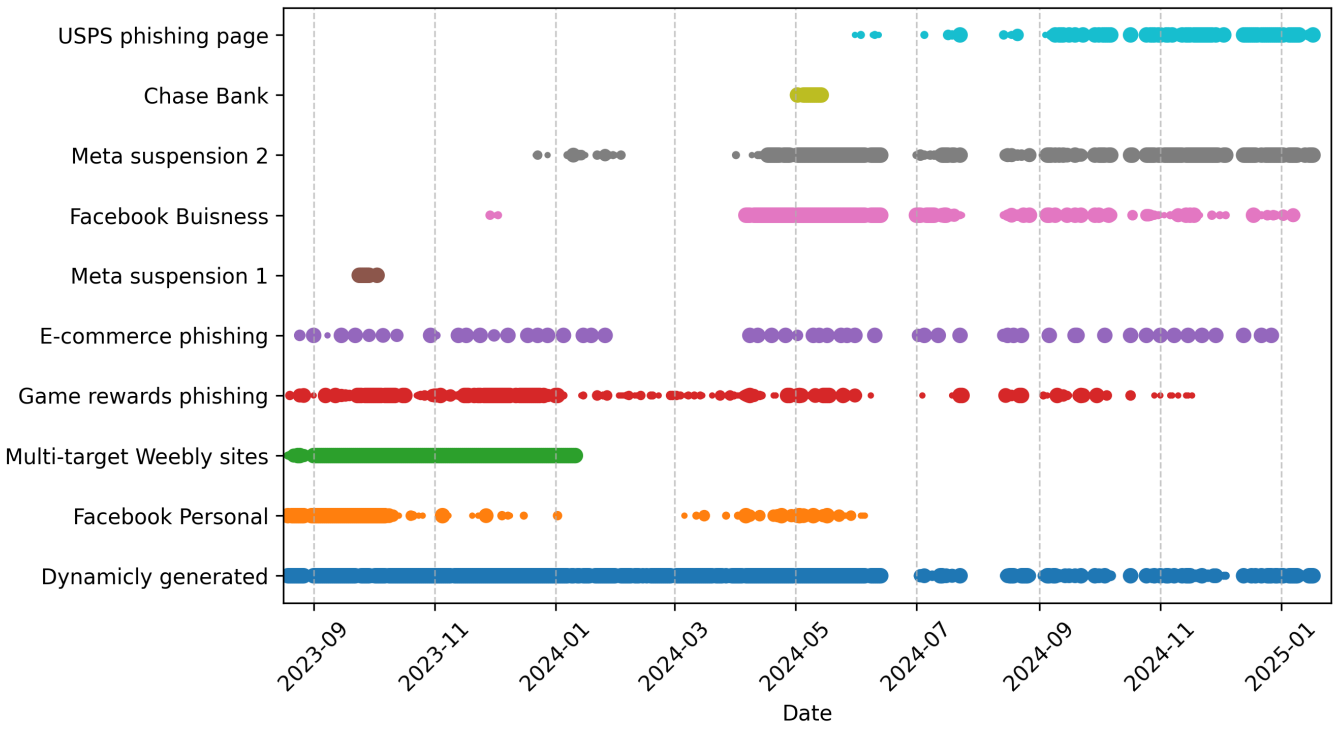


Fig. 4: Timeline of the top 10 clusters

Finding 10: *Pop-UP APIs are on a decline*. We see only 104 cluster (1,323 pages) call out to pop-up requesting APIs. Among these, the most popular was Geolocation.getCurrentPosition (55 clusters). This API, while requiring a pop-up to interact with, can also play a crutial role in cloaking, as the results are not masked by any VPN or proxy.

We observe a smaller fraction of the ecosystem (16 clusters, 148 pages) then [56] employ this cloaking techniques, especially when it comes to triggering a notification pop-up to verify user interaction. This could be a result of Firefox, citing low engagement with the notifications, started requiring user interaction to trigger the popup[34] at the end of November 2019, when crawlphish's data collection ended. Chrome has since discussed modifying the notification API to make the request less disruptive to the user experience[3]. The lack of pop-up requests could also be explained by our **Finding-XX** regarding usage of Fetch and XMLHttpRequest or by the overwhelming amount of the pages (67%) registering at least one HTMLElement event handler, which would be classified as a *Click-through* by Crawlphish's taxonomy. We report a full breakdown of APIs related to the crawlphish categorization of client-side cloaking in Table IV.

Finding 11: *Mouse Detection API calls are specific to a small group of clusters*. While supported by most modern browsers, we see a very rare use of Mouse Detection APIs. Only 35 clusters employ mouse detection related APIs. Two of these clusters[6] are from an open-source phishing kit, leaveraging botguard, from github which was last updates in 2017[7]. We see these clusters deploy across 17 unique domains, starting from 2023-10-07 all the way to 2024-07-19.

Finding 12: *Phishing pages make network requests using javascript before any user interaction*. Client-side IP reputation check is **Finding 13**: *The phishing ecosystem consists of clusters that utelize both cutting edge, experimental browser APIs, and extreamly depricated APIs*. We find 421 clusters (8270 pages) that utelize, *NavigatorUAData.getHighEntropyValues* for fingerprinting and *Keyboard.lock* to restrict user input, APIs not fully supported by Firefox and Safari. We identified 10 clusters across XXX pages that used *Scheduling.isInputPending*, however, upon closer inspection, these were not pages using a novel kit, but rather pages that used Google Sheets in order to construct their landing page. On the other hand, 25% of the clusters spanning (47,001) pages use a depricated API.

VI. Discussion (1 page)

A. Why clusters?

As we have shown in the examples in this paper, phishing feeds can be a noisy source of data for study the overall ecosystem. From e-commers pages, to mass spammed USPS and EZ-parking phishing pages, differentiated between handful of pages of verying sizes becomes crutial for efficiency. In this paper, we demonstrate that browser API usage, can closely proxy the phishing kit (providing commonality for the page's

[6]Split due to infrastructure insability causing crashes in easy crawls

[7]https://github.com/ashanahw/Gmail_Phishing

Brand	Phishing pages observed	Avg % Similar	Std Dev (%)	Perfect Subset	50%+ Match
Facebook	338,686	11.6	9.6	0	24,039
USPS	66,692	10.3	10.5	0	53
Meta	51,969	12.4	7	0	2,605
Microsoft	3,768	11.8	8.7	0	419
IRS	1,207	11.5	9	0	61

TABLE III: Comparison of phishing page metrics across different brands

origin) as best, and group via common client-side techniques as worst.

Furthermore, kit identification lets you infer server-side behavior about a particular page as long as a kit is identified and analyzed for one of the pages. As established by Oest*et al.*, Phishing kits thrive on including up-to-date IP blocklists. However, this blocklist is not always included in the client-side code. Being able to link pages to the same kit without any anti-evasion techniques like dynamic execution will allow better triaging of failed detections from phishing feed crawlers, something prior work established can be done by prior profiling[PriorProfiling].

B. What makes a kit identifiable?

Sets of browser APIs used is not the most granual method to describe pages, however, without with some many degrees of freedom in choises, the most sophisicated the phishing kit becomes, the easier it is to spot by just the browser APIs it uses. In plain javascript, without browser APIs, a page can not reach out to a C2 serve, gather browser finterprints, dybamically generate or cloak page contents, or request browser pop-ups without calling out to a browser API. In the case of the pages clustered in Figure-3, the APIs *Keyboard.lock*, *HTMLDocument.onkeydown* for keyboard locking, *Window.atob* for obfuscation, and a handful of fingerprint APIs and DOM APIs for dynamic content generation, set these pages apart from the other clusters.

Finding 14: *Phishing pages vary wildly from the brand that they are mimicking*. We collect browser API traces from the login pages of Facebook, USPS, Meta, Microsoft, and IRS and compare them to their phishing counterparts. The average similarity of the APIs executed by the phishing page and the original page is 11%, indicating that browser APIs do not relate to the target page. We found no pages where the original page's API set was a subset of the phishing page's API set, and in 5% of the cases, the phishing page executed at least half of the APIs from the original page. We report per-brand findings in Table- III and note that the least similar brand was USPS, which could be the result of USPS phishing pages being multi-stage pages requiring user interaction and targeting credit card information[https://isc.sans.edu/diary/30078]

This, however, does not indicate to use that browser API usage along is a good indicator of malicousness. All of the tehcniques described in Section-IV are common throughout the web, however, phishing pages are singular in their purpose, and their choise to engage in these techniques is what sets them apart from one another.

C. Dynamic analysis combats evasions

While dynamic analysis suffers from drawbacks we will discuss in Section-VII, in javascript, it allows us to effectively sidesteps any sophisicated obfuscation techniques. Even something something as neuoused as an AES encrypted script will show up in VisisbleV8 as long as the bahavior is triggered. While not incorporated by our work, prior-work has shown forced execution [41], [56] to be an effective way of ensuring all behaviors are extracted by dynamic analysis.

VII. Limitations and Future Work (.5 page)

A. Veriance

We collect API traces from these pages by remaining on these pages for 15 seconds. While this is in line with what prior work employs, lack of force execution, page interactions, and differeint system loads between runs, introduce noise into our measurement. As highlighted by other work [50], modern phishing pages include a multi-step user experience, and future work can aim extrapolate the techniques used by pages at different stages via intrumented crawlers that interact with the pages.

B. Bias sampling

While most phishing pages have been shown to race the clock against being listed on a blocklist eventually, prior work has shown that these blocklists can have blind spots. Prior work has also shown a bias towards certain targets in different feeds[Citation needed]. While we follow best practices in diversifying our sources for phishing pages,

C. Limited Kit sophistication

The phishing kits in this paper were collected using Kit-Phishr; while this has been done in prior research, this requires a misconfiguration on the side of the deployer, thus the bias towards PHP, as it is relatively easy to miss-configure apache to allow downloading of the zip file if it is placed in the document root. However, modern backend web technologies, like ExpressJS, Flask, and even Go's built-in HTTP server, would nullify this weakness. The dataset of diverse phishing kits is hard to come by. It could only be obtained through gaining trust in communities where they are freely shared (i.e., Telegram groups, forums, or LinkedIn), direct purchasing, or honeypot setup similar to [19] but one that would enable deployment of any kits.

VIII. Conclusion (.25 page)

In this paper, we provide a workflow for researches and analysts to automatically differentiate between a collections of phishing pages, based on common underlying kits, or shared techniques if the behaviors are too generic. We show an accuracy of 97% against a dataset of pages and kits collected from the wild. With a curated mapping of techniques to browser APIs and over 500,000 pages in which we identify 11,377 clusters, we explore what techniques are universal, widespread across kits, or kit-specific. We find that the ecosystem as a whole, as adapted some form of fingerprinting,

References

[1] APWG | Phishing Activity Trends Reports. https://apwg.org/trendsreports/.

[2] cybercdh/kitphishr: A tool designed to hunt for phishing kit source code.

[3] Introducing quieter permission UI for notifications.

[4] Eval() - JavaScript | MDN. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval, January 2025.

[5] Unit 42. Threat Actor Groups Tracked by Palo Alto Networks Unit 42, June 2024.

[6] Sahar Abdelnabi, Katharina Krombholz, and Mario Fritz. VisualPhishNet: Zero-day phishing website detection by visual similarity. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, pages 1681–1698. Association for Computing Machinery.

[7] Bhupendra Acharya and Phani Vadrevu. {PhishPrint}: Evading phishing detection crawlers by prior profiling. pages 3775–3792.

[8] Anti-Phishing Working Group (APWG). eCrime Exchange (eCX). https://apwg.org/, 2025.

[9] Hugo Bijmans, Tim Booij, Anneke Schwedersky, Aria Nedgabat, and Rolf van Wegberg. Catching phishers by their bait: Investigating the dutch phishing landscape through phishing kit detection. pages 3757–3774.

[10] Felipe Castaño, Eduardo Fidalgo Fernañdez, Rocío Alaiz-Rodríguez, and Enrique Alegre. PhiKitA: Phishing kit attacks dataset for phishing websites identification. 11:40779–40789. Conference Name: IEEE Access.

[11] Daiki Chiba, Hiroki Nakano, and Takashi Koide. Domainlynx: Leveraging large language models for enhanced domain squatting detection. *ArXiv*, abs/2410.02095, 2024.

[12] Cisco Talos Intelligence Group (Talos). Phishtank. https://phishtank.org/, 2025.

[13] Marco Cova, Christopher Kruegel, and Giovanni Vigna. There is no free phish: an analysis of "free" and live phishing kits. In *Proceedings of the 2nd Conference on USENIX Workshop on Offensive Technologies*, WOOT'08, USA, 2008. USENIX Association.

[14] Dinil Mon Divakaran and Adam Oest. Phishing Detection Leveraging Machine Learning and Deep Learning: A Review. *IEEE Security & Privacy*, 20(5):86–95, September 2022.

[15] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of ACM CCS 2016*, 2016.

[16] Edward B Fowlkes and Colin L Mallows. A method for comparing two hierarchical clusterings. *Journal of the American statistical association*, 78(383):553–569, 1983.

[17] Google Inc. Catapult. https://chromium.googlesource.com/catapult/, 2025.

[18] Google Inc. Puppeteer. https://pptr.dev/, 2025.

[19] Xiao Han, Nizar Kheir, and Davide Balzarotti. PhishEye: Live Monitoring of Sandboxed Phishing Kits. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1402–1413, New York, NY, USA, October 2016. Association for Computing Machinery.

[20] Microsoft Threat Intelligence. Franken-phish: TodayZoo built from other phishing kits. https://www.microsoft.com/en-us/security/blog/2021/10/21/franken-phish-todayzoo-built-from-other-phishing-kits/, October 2021.

[21] Jordan Jueckstock and Alexandros Kapravelos. VisibleV8: In-browser Monitoring of JavaScript in the Wild. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2019.

[22] Koide, Takashi and Fukushi, Naoki and Nakano, Hiroki and Chiba, Daiki. Phishreplicant: A language model-based approach to detect generated squatting domain names. In *Proceedings of the 39th Annual Computer Security Applications Conference*, ACSAC '23, page 1–13, New York, NY, USA, 2023. Association for Computing Machinery.

[23] Brian Kondracki, Babak Amin Azad, Oleksii Starov, and Nick Nikiforakis. Catching Transparent Phish: Analyzing and Detecting MITM Phishing Toolkits. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, pages 36–50, New York, NY, USA, November 2021. Association for Computing Machinery.

[24] Hung Le, Quang Pham, Doyen Sahoo, and Steven C. H. Hoi. Urlnet: Learning a url representation with deep learning for malicious url detection. *ArXiv*, abs/1802.03162, 2018.

[25] Woonghee Lee, Junbeom Hur, and Doowon Kim. Beneath the phishing scripts: A script-level analysis of phishing kits and their impact on real-world phishing websites. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, pages 856–872. ACM.

[26] Kyungchan Lim, Jaehwan Park, and Doowon Kim. Phishing vs. legit: Comparative analysis of client-side resources of phishing and target brand websites. In *Proceedings of the ACM Web Conference 2024*, WWW '24, New York, NY, USA, 2024. Association for Computing Machinery.

[27] Xu Lin, Panagiotis Ilia, Saumya Solanki, and Jason Polakis. Phish in sheep's clothing: Exploring the authentication pitfalls of browser fingerprinting. pages 1651–1668.

[28] Xu Lin, Panagiotis Ilia, Saumya Solanki, and Jason Polakis. Phish in Sheep's Clothing: Exploring the Authentication Pitfalls of Browser Fingerprinting. pages 1651–1668.

[29] Ruofan Liu, Yun Lin, Xianglin Yang, Siang Hwee Ng, Dinil Mon Divakaran, and Jin Song Dong. Inferring phishing intention via webpage appearance and dynamics: A deep vision based approach. pages 1633–1650.

[30] Ruofan Liu, Yun Lin, Yifan Zhang, Penn Han Lee, and Jin Song Dong. Knowledge expansion and counterfactual interaction for {Reference-Based} phishing detection. pages 4139–4156.

[31] Heather McCalley, Brad Wardman, and Gary Warner. Analysis of Back-Doored Phishing Kits. In Gilbert Peterson and Sujeet Shenoi, editors, *Advances in Digital Forensics VII*, pages 155–168, Berlin, Heidelberg, 2011. Springer.

[32] Ettore Merlo, Mathieu Margier, Guy-Vincent Jourdan, and Iosif-Viorel Onut. Phishing kits source code similarity distribution: A case study. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 983–994. ISSN: 1534-5351.

[33] Mitchell Krog and Nissar Chababy. PhishingDB. https://github.com/Phishing-Database/Phishing.Database, 2025.

[34] Mozilla. Restricting Notification Permission Prompts in Firefox. https://blog.mozilla.org/futurereleases/2019/11/04/restricting-notification-permission-prompts-in-firefox, November 2019.

[35] Aleksandr Nahapetyan, Sathvik Prasad, Kevin Childs, Adam Oest, Yeganeh Ladwig, Alexandros Kapravelos, and Bradley Reaves. On sms phishing tactics and infrastructure. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 169–169. IEEE Computer Society, 2024.

[36] Adam Oest, Yeganeh Safaei, Adam Doupe, Gail-Joon Ahn, Brad Wardman, and Kevin Tyers. PhishFarm: A Scalable Framework for Measuring the Effectiveness of Evasion Techniques against Browser Phishing Blacklists. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1344–1361, San Francisco, CA, USA, May 2019. IEEE.

[37] Adam Oest, Yeganeh Safei, Adam Doupé, Gail-Joon Ahn, Brad Wardman, and Gary Warner. Inside a phisher's mind: Understanding the anti-phishing ecosystem through phishing kit analysis. In *2018 APWG Symposium on Electronic Crime Research (eCrime)*, pages 1–12. ISSN: 2159-1245.

[38] Adam Oest, Penghui Zhang, Eric Nunes, Jakub Burgis, Ali Zand, Kurt Thomas, and Adam Doupé. Sunrise to sunset: Analyzing the end-to-end life cycle and effectiveness of phishing attacks at scale.

[39] Berstend on Github. Puppeteer stealth plugin. https://www.npmjs.com/package/puppeteer-extra-plugin-stealth, 2025.

[40] OpenPhish. OpenPhish. https://www.openphish.com/, 2025.

[41] Nikolaos Pantelaios and Alexandros Kapravelos. FV8: A Forced Execution JavaScript Engine for Detecting Evasive Techniques. In *Proceedings of the USENIX Security Symposium*, August 2024.

[42] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas,

A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[43] Ch. Chakradhara Rao, A.V.T. Raghav Ramana, and B. Sowmya. Detection of phishing websites using hybrid model. 2018.

[44] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Conference on Empirical Methods in Natural Language Processing*, 2007.

[45] Iskander Sanchez-Rola, Leyla Bilge, Davide Balzarotti, Armin Buescher, and Petros Efstathopoulos. Rods with laser beams: Understanding browser fingerprinting on phishing pages. pages 4157–4173.

[46] Shaown Sarker, Jordan Jueckstock, and Alexandros Kapravelos. Hiding in Plain Site: Detecting JavaScript Obfuscation through Concealed Browser API Usage. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, page 648–661, October 2020.

[47] SecurityTrails. Urlscan. https://urlscan.io/, 2025.

[48] Hossein Shirazi, Bruhadeshwar Bezawada, and Indrakshi Ray. "kn0w thy doma1n name": Unbiased phishing detection using domain name based features. *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*, 2018.

[49] Junhua Su and Alexandros Kapravelos. Automatic discovery of emerging browser fingerprinting techniques. In *Proceedings of the ACM Web Conference 2023*, WWW '23, page 2178–2188, New York, NY, USA, 2023. Association for Computing Machinery.

[50] Karthika Subramani, William Melicher, Oleksii Starov, Phani Vadrevu, and Roberto Perdisci. PhishInPatterns: measuring elicited user interactions at scale on phishing websites. In *Proceedings of the 22nd ACM Internet Measurement Conference*, IMC '22, pages 589–604. Association for Computing Machinery.

[51] Xiwen Teoh, Yun Lin, Ruofan Liu, Zhiyong Huang, and Jin Song Dong. {PhishDecloaker}: Detecting {CAPTCHA-cloaked} phishing websites via hybrid vision-based interactive models. pages 505–522.

[52] Charles Truong, Laurent Oudre, and Nicolas Vayatis. Selective review of offline change point detection methods. *Signal Processing*, 167:107299, 2020.

[53] Jonas Tzschoppe and Hans Löhr. Browser-in-the-middle - evaluation of a modern approach to phishing. In *Proceedings of the 16th European Workshop on System Security*, EUROSEC '23, pages 15–20. Association for Computing Machinery.

[54] Rakesh M. Verma and Avisha Das. What's in a url: Fast feature extraction and malicious url detection. *Proceedings of the 3rd ACM on International Workshop on Security And Privacy Analytics*, 2017.

[55] 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. CrawlPhish: Large-scale analysis of client-side cloaking techniques in phishing. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1109–1124. ISSN: 2375-1207.

[56] 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. Crawlphish: Large-scale analysis of client-side cloaking techniques in phishing. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1109–1124, 2021.

[57] Penghui Zhang, Zhibo Sun, Sukwha Kyung, Hans Walter Behrens, Zion Leonahenahe Basque, Haehyun Cho, Adam Oest, Ruoyu Wang, Tiffany Bao, Yan Shoshitaishvili, Gail-Joon Ahn, and Adam Doupé. I'm SPARTACUS, No, I'm SPARTACUS: Proactively Protecting Users from Phishing by Intentionally Triggering Cloaking Behavior. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 3165–3179. ACM.

APPENDIX A

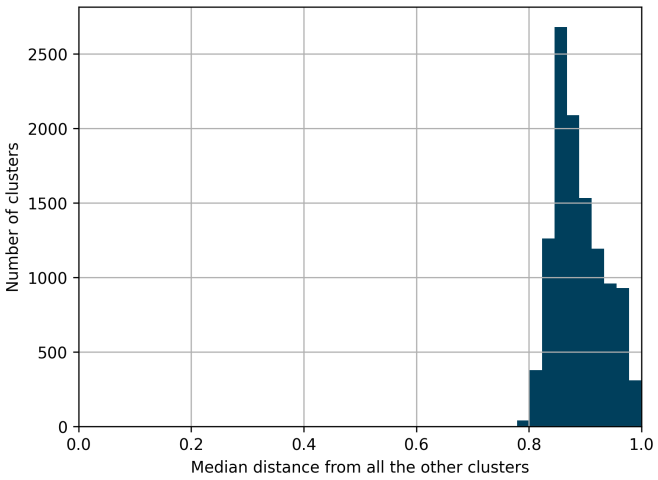


Fig. 5: Histogram of the Jaccard index based simularity between any pair of pages belonging to the same kit, versus from different kits

TABLE IV: Summary of all API calls that match to a category identified by Crawlphish

Category	Clusters	Pages
User-Interaction		
Pop-up (Total)	104	1,323
Accelerometer	7	63
Clipboard.readText	1	2
Geolocation.getCurrentPosition	55	479
Gyroscope	4	10
MediaDevices.getUserMedia	5	55
Notification.requestPermission	16	148
Window.alert	22	581
Mouse	35	62
DomEvents	10,402	362,700
Fingerprint		
Total	9,077	287,983
Navigator.userAgent	8,641	266,126
HTMLDocument.cookie	4,664	124,110
HTMLDocument.referrer	0	0
Bot Detection		
Timing	XXX	XXX